applied sciences

MDPI

Article

Low Earth Orbit Satellite Network Routing Algorithm Based on Graph Neural Networks and Deep Q-Network

Yuanji Shi *, Weian Wang, Xiaorong Zhu and Hongbo Zhu

College of Telecommunications and Information Engineering, Nanjing University of Posts and Telecommunications, Nanjing 210003, China; 1223013707@njupt.edu.cn (W.W.); xrzhu@njupt.edu.cn (X.Z.); zhuhb@njupt.edu.cn (H.Z.)
* Correspondence: shiyuanji@nj-ict.cn

Abstract: Low Earth orbit (LEO) satellite networks are characterized by rapid topological changes, numerous network nodes and varying states of node resource constraints, which have resulted in traditional routing algorithms no longer being suitable for LEO satellite network routing. Therefore, this paper proposes an inductive learning architecture based on Graph Sample and Aggregate (GraphSAGE), which can significantly reduce the number of topology nodes to be trained, thereby reducing the computational complexity of the nodes. Then deep reinforcement learning (DRL) is employed for the continuous learning optimization of routing algorithms, and its generalization is improved by selecting GraphSAGE to construct the DRL agent. In the proposed graph neural-network-based routing optimization algorithm for LEO satellite networks, each Deep Q-Network (DQN) agent independently generates the hidden states of the nodes through the GraphSAGE model and uses them as inputs to the DRL model to make routing decisions. After a simulation and comparison, the proposed algorithm not only improves the overall network throughput, but also reduces the average end-to-end delay. The average throughput of the proposed algorithm increases by 29.47% and 18.42% compared to that of Dijkstra and the DQN, respectively. The average end-to-end delay is reduced by 39.76% and 15.29%, respectively, and can also adapt to changing topologies.

Keywords: LEO satellite; routing; graph neural network; DQN

Citation: Shi, Y.; Wang, W.; Zhu, X.; Zhu, H. Low Earth Orbit Satellite Network Routing Algorithm Based on Graph Neural Networks and Deep Q-Network. *Appl. Sci.* **2024**, *14*, 3840. https://doi.org/10.3390/app14093840

Academic Editor: Mario Gai

Received: 14 March 2024
Revised: 20 April 2024
Accepted: 27 April 2024
Published: 30 April 2024

1. Introduction

In recent years, the development of LEO satellite networks has attracted more and more attention, and LEO network constellations are being actively built around the world, including OneWeb, Telsat and Starlink. Compared with the geosynchronous Earth orbit (GEO) satellite network system, the satellites in the LEO satellite communication system are located in low Earth orbit, resulting in lower levels of transmission latency, better channel conditions and higher transmission rates, which will greatly enhance the user experience. However, these satellites are always in high-speed motion relative to the ground, so the number of satellites deployed will be much larger than the number of GEO satellites if global coverage is required using LEO satellites. With the increase in the number of satellites and the high-speed movement of satellites, the LEO satellite network structure will also become extremely complex. In order to achieve low levels of latency and a high-throughput performance in the case of a complex network structure, an efficient routing algorithm for LEO satellite networks becomes very important.

The complex LEO network structure poses great challenges for routing algorithms. The high-speed movement of satellites and the large number of satellites have led to the instability of satellite network topology and the inability to make accurate predictions in advance.

There is not much current work on routing algorithms based on LEO satellite networks, as most studies focus on self-organizing networks. Ref. [1] has used a combination of graph neural networks and deep learning for LEO routing algorithms, but the graph neural

network algorithm (GNN) does not take into account the computational complexity caused by the large number of nodes in the LEO satellite network and the changing topology when using algorithms. In LEO satellite networks, due to their large number of nodes, we need to consider how to perform distributed training on large graph data and need to be able to learn about nodes that have not appeared before. P. Zuo et al. [2] relies only on deep learning to consider the design of routing algorithms. However, they do not address the challenge of adapting to the dynamic topology of LEO satellite networks. Many scholars have studied routing algorithms for self-organized networks and refer to the routing algorithms for LEO satellite networks, but they do not consider the characteristics of LEO satellite nodes, potential unpredictable topological changes, and instances of limited computational resources. Most of them take into account the state of the nodes while finding the shortest paths. As in [3], only the congestion state of the nodes and the channel quality are considered using deep learning. Ref. [4] introduces a graph neural network for the design of a routing algorithm using deep learning, but its convergence method only considers the characteristics of wireless sensor network nodes. The literature does not consider that if the computing resources of the nodes are limited, it is difficult to complete the whole topology computation. Ref. [5] uses a deep learning approach to design routing algorithms for wireless self-organizing networks, mainly considering the effect of QoS. However, this study also fails to consider the effect of rapid topological changes. Ref. [6] applies a deep learning approach to the design of routing algorithms for ad hoc networks, mainly considering end-to-end delay. Again, this study does not consider the effect of rapid topological changes.

In summary, it is necessary to design a routing algorithm that can use distributed training to reduce the number of training nodes and computational complexity and can continuously learn to optimize the paths for the node state and link state.

The intelligent routing algorithm based on deep learning is an algorithm that utilizes deep learning techniques to make decisions on routing in a network. It automates the selection of appropriate routing paths for the efficient transmission of network traffic by learning the data flow patterns and laws in the network. In the intelligent routing algorithm based on deep learning, the model first needs to be trained with a large amount of data, which include the network topological information, traffic load conditions, congestion information to learn patterns and laws of the network. Then, the trained deep learning model is utilized to make routing decisions.

Compared with traditional routing algorithms, intelligent routing algorithms based on deep learning are able to monitor network traffic in real time and improve the overall network performance [7]. However, when the network topology changes, it needs to readjust the training labels to output the correct paths, so the traditional deep learning scheme cannot guarantee the correctness of the output paths. Moreover, the traditional deep learning scheme is not scalable. Even though the model is trained with the readjusted input data when the input network topology changes, this may cause a high processing latency.

As an important branch of machine learning, DRL [8,9] differs from supervised and unsupervised learning and involves learning interactively with the environment. Unlike traditional reinforcement learning methods, DRL uses deep neural networks (DNNs) instead of tables and expresses policies with function approximations. In this way, DRL avoids the dimensionality problem that traditional tabular reinforcement learning faces when dealing with complex real-world problems.

Although trained DRL agents can significantly improve the routing optimization problem, they cannot work effectively in unknown network topologies. By analyzing the structure of DRL agents, the reason for this phenomenon can be found. In the case of a classical DQN, the inputs and dimensions of the model are determined by the actual network size (e.g., the size of the network topology). After the training is finished, the dimensions of the model inputs and outputs are fixed. Therefore, when facing networks of different sizes, the model is likely to receive inputs with anomalous dimension sizes. Although we can change the input dimensions by cutting or padding, it will destroy

the potential topological information in the matrix. In addition, computer networks are essentially graphs, and existing methods that use neural networks to process state matrices are unable to learn the information in graph structures, which limits the performance of DRL in new networks. In conclusion, although DRL performs well in routing optimization, it cannot perform relational reasoning or graph structure generalizations to operate effectively in new network environments.

A GNN is a neural network structure that can effectively deal with irregular topological information [10–12]. The basic principle of a GNN is to express and infer the nodes' characteristics through their local information and the relationship between neighboring nodes. Each node has an initial feature vector, which can be updated and improved through information interactions and aggregation with neighboring nodes. A GNN can be used to obtain global graph characterization and prediction results through multiple rounds of information transfers and feature updates. A graph convolutional neural network (GCN) is better able to characterize network structures and features, which can automatically extract the hidden and complex information patterns in a graph using a graph convolution operator. However, the GCN's shortcoming is that it only acquires the hidden features of all of the nodes and does not have the ability to scale.

The GraphSAGE [13] model is an inductive learning architecture that uses information from the current node and neighboring nodes to form feature vectors by aggregation. GraphSAGE is able to represent feature vectors for any node through an aggregation function so that it has the ability to generalize, which is different from direct push learning. Therefore, in this paper we chose GraphSAGE to build the DRL agent to improve its generalization. And each DQN agent independently generates the hidden states of the nodes through the GraphSAGE model and uses them as inputs to the DRL model to make routing decisions.

In this paper, the GNN-DRL algorithm is proposed to adapt to dynamic LEO networks with frequent topological changes and to solve the network congestion problem, which not only improves the overall network throughput but also reduces the average end-to-end delay compared with the traditional routing algorithm. To solve the fluency problem of the network topology changing and reduce the computational complexity of the nodes, the graph neural networks are used to learn the relationships between the graph elements in the network topology as well as the node composition rules. Then deep reinforcement learning algorithms are used to make routing decisions. To reduce the computational complexity of the nodes, GraphSAGE reduces the number of topology nodes to be trained.

The remainder of this paper is structured as follows: The algorithmic model is presented in Section 2. Section 3 describes the progress of the feature extraction with GraphSAGE. Section 4 describes the routing policy for the DQN agent. The evaluation results are shown in Section 5. Section 6 concludes this paper.

2. System Model

According to the dynamic characteristics of low-orbit satellite network topology, in this paper, we design the GNN-DRL algorithm, which is based on the following assumptions:

1. According to the interval of t_0, the time is divided into n time periods $(t_1, t_2, t_3 \ldots t_n)$, and the physical topology of the constellation remains unchanged in the time period between two adjacent moments;
2. It is assumed that the propagation delay between the two satellites in different orbits is almost the same as that in the same orbit.

Authors of studies in the literature [14] believe that the selection of the location and number of ground stations is a key issue for low-orbit satellite networks, which directly affects their construction. It is mentioned that one of the criteria for determining the location of ground stations is the number of satellites (e.g., m) that a ground station can connect to. So, for the algorithm model designed in this paper, the ground station will also be an important factor to be considered. The agent of the algorithm proposed in this paper is placed on the ground station, which is responsible for sending the final execution

action (routing final decision) to m satellites. The network composed by m satellites that is managed by each ground station is called the regional network in this paper. Different regional networks are connected by selecting an inter-satellite link with stable (as in, the same orbital plane) and small load between regional networks to form the final LEO satellite network.

Firstly, the problem of dynamic topological changes in low-orbit satellite networks has two parts: topological changes caused by satellite node movement and topological changes caused by satellite state or link changes. Then, solving the routing problem of topological changes caused by the movement of satellite nodes is equivalent to solving the routing problem of a fixed topology at different times ($t_1, t_2, t_3 \ldots t_n$) through the above assumptions. In the time period in which the topology is unchanged, the routing problem with link state and satellite node state changes is equivalent to a routing problem of a hierarchical network with a fixed topology, but link state and node state may change through the strategy of network classification according to the ground station. In order to make the problem easier to solve, hierarchical networks with the characteristics of fixed topology but possible changes in link state and node state are considered as equivalent to NSFNET networks with the same characteristics.

The LEO satellite network can be represented as a graph $G = (\mathrm{V}, \mathrm{E})$ during t_0 period, where V and E denote the set of LEO satellite nodes and edges between them. Then the routing problem of the LEO satellite network can be defined as follows: given $G = (\mathrm{V}, \mathrm{E})$, the source node *src*, the destination node *dst* and the traffic flow demand *bw*, find a set of paths in order to forward the data from the source to the destination. And when the inter-satellite link changes, E will show the changes in the links. The goal is to minimize the network end-to-end delay while increasing the network throughput, whether the network topology is changing or not.

It is found that its future state only depends on the current state by analyzing the LEO satellite network routing problem, which is a typical Markov decision process. Therefore, all node states can be directly input into DRL to seek the maximum reward to find the optimal path. But DRL cannot learn topological information when dealing with routing optimization problem. Once one or more nodes fail for some reason, the network topology will change. The routing algorithms based on DRL do not work well when the network topology changes. This is because DRL algorithms mainly focus on the state information of individual network nodes and ignore the impact of selecting the routing for the individual node on the whole network topology. The DRL algorithms also do not consider the network as a whole, so they cannot utilize the global graph of the network state to dynamically adapt to changes in the network. Therefore, in order to better optimize routing, it is necessary to consider network topological information and node state information comprehensively and analyze the network as a whole to dynamically adapt to changes in the network. Therefore, this paper proposes the GNN-DRL algorithm, which is a network routing optimization architecture based on graph neural networks and deep reinforcement learning. First of all, the graph neural networks are used to learn the relationships between graph elements in the network topology as well as the node composition rules. Then deep reinforcement learning algorithms are used to make routing decisions. The input state of DRL agent is expected to include not only the current node state and link state, but also the overall network state of the node according to the dynamic characteristics of LEO satellite network. Then we need to aggregate network information through an aggregation algorithm. However, due to the large number of nodes in LEO satellite network, if the common GCN is used, it will need a large number of calculations and cannot complete inductive representation learning. So, this paper uses GraphSAGE to aggregate the attributes of the node itself and its limited number of neighbors. The algorithm model diagram is shown in Figure 1, and the algorithm has following processes:

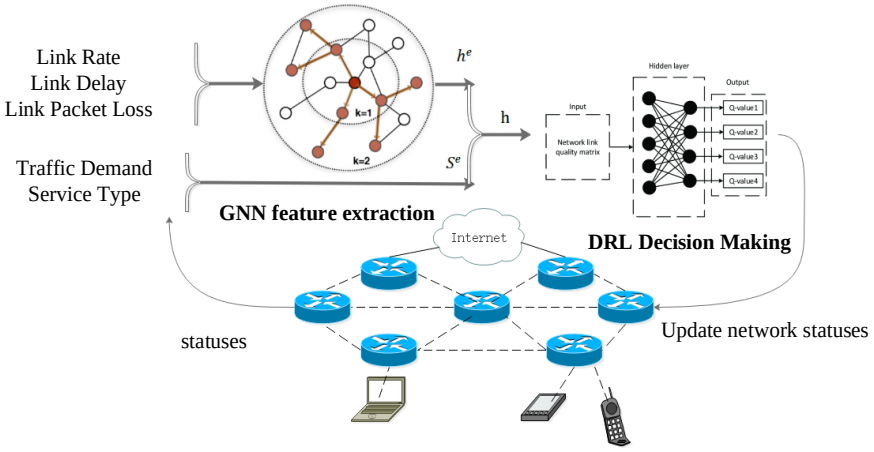

Figure 1. Algorithm diagram based on GNN-DRL.

Step1: For the current topology from t_n to $t_n + t_0$, a graph neural network is utilized to learn the network topological information and perform information extraction, through which the network state is represented.

Step2: The network state outputted by the graph neural network combined with the user state is input into the DRL to make routing decisions and update the network state. And the best decision-making action is selected based on the node user state as well as the network state to maximize the discounted cumulative rewards. And the policy parameters θ are updated every γ ($t_0 = n\gamma$).

2.1. Network Model

The algorithms proposed in this paper not only need to adapt to the dynamic network with changing network topology, but also need to obtain better network performance and lower average end-to-end network delay. Therefore, it is necessary to comprehensively study the network topology as well as the network characteristics of links and nodes. However, GraphSAGE can only deal with the network topology and the network characteristics of nodes and cannot analyze the network characteristics of links. To solve this problem, the graph structure is reconstructed by aggregating link information into nodes.

The specific implementation of this is shown in Algorithm 1 (a reconfiguration map of data-specific processes). The input is the network topology information, which is represented by an undirected graph $G = (\mathrm{V}, \mathrm{E})$, where $\mathrm{V} = \{v_i\}_{i=1:N}$, $\mathrm{E} = \{(e_k, r_k, s_k)\}_{k=1:L}$ denote the set of edges and the set of nodes, respectively. The variable e_k on each edge is a one-dimensional vector containing information such as link capacity and delay. The output is a new graph structure $G(\mathrm{V}', \mathrm{E}')$ with the features of the edges related to this node aggregated in v_i'.

Algorithm 1. Reconfiguring Graph Data

1: Initializing the network $G = (\mathrm{V}, \mathrm{E})$
2: **for** $k \in \{1 \cdots L\}$ **do**
3: $e_k' \leftarrow \varnothing^e(e_k, r_k, s_k)$ //Update Edge Properties
4: **end for**
5: **for** $i \in \{1 \cdots N\}$ **do**
6: $E_i = \left\{ \left(e_k', r^k, s^k \right) \right\}_{r^k = i, k = 1:L}$ //Find edges adjacent to node i
7: $\bar{e}_i \leftarrow \rho(E_i)$ //Aggregate edge attributes per node
8: $v_i' \leftarrow \varnothing^v(\bar{e}_i, v_i)$ //Update node attributes
9: **end for**
10 : **return** $G(V', E')$

The network characteristics of each node are represented by a feature vector $x_i (i = 0, 1 \cdots N-1)$, $x_i = [q_i, v_i']$, where q_i denotes the traffic flow. v_i' indicates the aggre-

gated characteristics of the edges related to this node, including information such as link capacity and delay.

Then the network feature matrix X of the whole network topology can be expressed as follows:

$$X = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix} \tag{1}$$

2.2. Markov Decision Model

In this subsection, the state space, the action space and the reward function are defined, respectively.

(1) State space

The network state is defined by link characteristics such as link delay, link capacity and current link utilization. These characteristics are stored in a fixed size vector filled with zero padding. At the beginning of a DRL event, a hidden feature vector representation of all nodes is generated by a graph neural network. When the DRL assigns a traffic demand to a specific $src - dst$ path (that is, a sequence of links in the network), the network state changes. And the user state (including the source node src, the destination node dst, and the traffic demand bw) is added in the current network state. The ultimate goal is to reduce the average end-to-end delay and increase the network throughput at the end of the event (i.e., when the DRL agent iterates over all traffic demands).

(2) Action space

The number of possible routing paths corresponding to each traffic demand (that is, $src - dst$ node pair) leads to a high-dimensional action space, even in small networks. It makes the routing problem for DRL agents complicated, as it requires estimating which of the executable actions will result in the smallest end-to-end delay in a long period of time. In other words, the minimum end-to-end delay can be achieved by finding the optimal routing path for each traffic demand. Furthermore, actions need to be defined in a way that is invariant to the alignment of edges and nodes to exploit the ability of GNNs to generalize over graphs (i.e., only link-level features rather than specific identifiers or labels).

In order to adapt to dynamic networks with frequent topological changes, GNN-DRL algorithm employs a distributed routing decision mechanism that assigns a routing decision to each node instead of pre-constructing routing paths. We assume that a node u_i with m neighboring nodes will make a routing decision in the time period $t \in \{1, 2, \cdots, T\}$. In the process of selecting the routing path, the node u_i with the packet needs to select a neighboring node as the next hop to forward the packet. In time period t, we define an action $a_t \in A$, where $A = \{b_1, b_2, \cdots, b_m\}$ is the set of neighboring nodes of the node u_i. The action a_t represents the selected neighbor node to forward the packet for the node u_i, i.e., the node u_i selects the neighboring node b_j to forward the packet.

(3) Reward function

The reward function represents the different immediate rewards from different routing selections. In this chapter, the reward function R of the DQN agent is related to the QoS parameters which are as follows: delay (L) and rate (W). The goal of each agent is to reduce the network delay and improve the overall throughput of the network in a long period of time. The reward function R is determined as follows:

$$R = \alpha \cdot W - \beta \cdot L. \tag{2}$$

where $\alpha, \beta \in [0, 1]$ is the adjustable weight determined by the routing policy.

In general, the strategy function $\pi(a|s) : S \times A \to [0,1]$ is the conditional probability distribution of the agent when it chooses action a in state s. The goal of the policy at moment τ is to maximize the reward R_τ accumulated in the historical experience as follows:

$$R_\tau = \sum_{k=0}^{\infty} \gamma^k R_{\tau+k}. \tag{3}$$

where $\gamma \in (0,1)$ is a discount factor that weighs the historical and current reward data for the agent. Then, the expected return value of the state s after executing action a in the DRL agent (following policy π) is defined as the state–action value function as follows:

$$Q^\pi(s,a) = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{\tau+k} | s_\tau = s, a_\tau = a\right]. \tag{4}$$

The goal of the DRL algorithm is to find an optimal strategy that maximizes the long-term cumulative returns obtained from historical experience as follows:

$$\pi^*(a|s) = \arg max Q^\pi(s,a). \tag{5}$$

3. Feature Extraction with GraphSAGE

The GraphSAGE model is used to generate a hidden feature vector representation of its own node for each DRL agent. The core idea of the GraphSAGE method is to aggregate the neighboring nodes of each node, and then the result of the aggregation is used to update the node's representation. The aggregation process of the GraphSAGE is to aggregate the embedded representations of neighboring nodes into a fixed-length vector by a learnable function based on the embedded representations of the neighboring nodes and the node's feature vector. This aggregation function can be averaging or maximization, etc., or a more complex function such as a multi-layer neural network.

The process of using the GraphSAGE model for aggregating and updating the features of neighboring nodes of a target node is shown in Figure 2, which consists of three main steps: sampling, aggregation and generating embedding.

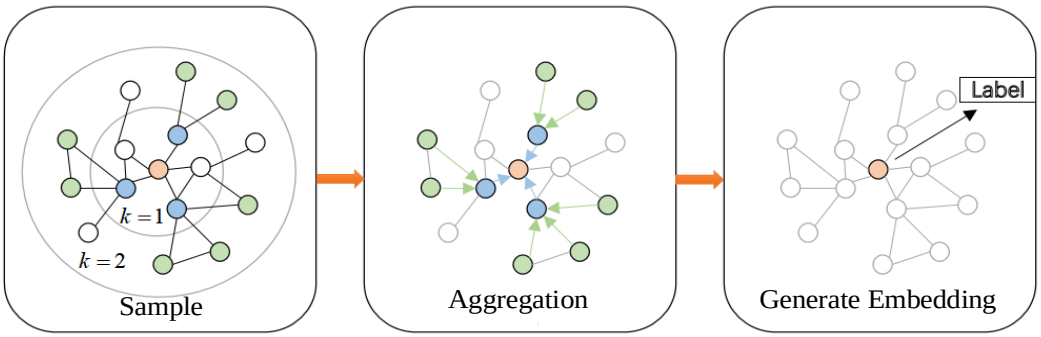

Figure 2. GraphSAGE node feature extraction process.

3.1. Neighborhood Node Sampling

For the target node v_i, the full sampling method is used to obtain a neighbor node set N_i containing q neighbor nodes, so the sampling number is q. q neighbor nodes are taken out at a certain time as the nodes for information aggregation. If the number of neighbors of a particular node is less than q, a resampling method with a put-back action is used. If the number of neighbors of a particular node is greater than q, a negative sampling method without a put-back action is used. This sampling method learns the feature based on neighbor relationships and generates self-observed hidden state features.

3.2. Neighboring Node Feature Aggregation

The GraphSAGE model has an L-layer network, and L usually takes the value of 2. When $L = 2$, the model performs two feature aggregations. During the aggregation process, the target node is able to receive the node information of its first-order neighbors and second-order neighbors. It aggregates the neighbor node information of each layer and updates its own node information through the aggregation function. The average aggregation function is used to obtain the aggregated features of the neighboring nodes in each layer in this paper. First of all, the feature representation of the target node in layer $k-1$ is concatenated with the aggregated features of the neighbor nodes in layer k. Then the mean value is calculated for each dimension of the vector. Finally, the final node representation is achieved by the nonlinear activation function.

The initial hidden state of each node consists of the node feature vector $\{x_0, x_1, \cdots x_{N-1}\}$ in the input layer, and the initial hidden state of the target node v_i is $h_i^0 = x_i$. First of all, the process to generate the hidden feature vector h_i^1 in the first layer of the network is shown in Equation (6):

$$h_i^1 = \sigma\Big(W^1 \cdot CONCAT\Big(h_i^0, AGGR\Big(h_j^0, \forall v_j \in N_i\Big)\Big)\Big). \tag{6}$$

where W is the parameter matrix and $\sigma(\cdot)$ is the nonlinear activation function. The neighbor aggregation feature $h_{N_i}^1$ of node v_i is obtained by aggregating the initial hidden states $h_j^0, \forall v_j \in N_i$ of all sampled neighbor nodes. The aggregation feature $h_{N_i}^1$ and the initial hidden state h_i^0 of v_i are concatenated together, and a nonlinear transformation is performed to generate the hidden feature representation h_i^1 of the target node v_i in the first network layer. The concatenated operation $CONCAT(\cdot)$ performs a linear superposition of the feature vectors using residual concatenation. And the aggregation function $AGGE(\cdot)$ uses mean value aggregation to calculate the mean value of the concatenated vectors. Finally, the final hidden feature vector is obtained using a nonlinear transformation. The mapping process between the middle two hidden layers is shown in Equation (7):

$$h_i^J = \sigma\Big(W^J \cdot CONCAT\Big(h_i^{J-1}, AGGE\Big(h_j^{J-1}, \forall v_j \in N_i\Big)\Big)\Big) \tag{7}$$

The hidden feature h_i^{J-1} of layer $J-1$ is concatenated and aggregated with the neighbor aggregation feature $h_{N_i}^J$ to obtain the hidden feature vector h_i^J of layer J for node v_i. The output of the last hidden layer is the final hidden feature vector h_i^L of node v_i. The GraphSAGE model can output the final hidden feature vectors of all nodes in the network. Since the GNN-DRL algorithm proposed in this section is a distributed DRL agent, only the final hidden feature vectors of all nodes are obtained without embedding process.

3.3. Learning Parameters

The parameter matrix is the core of GraphSAGE when the aggregation of node feature vectors is calculated, which contains the mapping relationship of information aggregation between the node feature vectors and neighboring features. So, this subsection explains the parameter learning process of the GNN-DRL algorithm. In the pre-training phase, the network parameters of the DQN model are fixed while the weight parameters of the GNN model are adjustable. Since the goal of model training is to maximize the cumulative reward R of the routing task, a gradient descent algorithm is used to calculate the gradient of the entire sample in order to learn the optimal weight parameter W of the GraphSAGE model, as shown in Equation (8):

$$W^t = W^{t-1} - \gamma \nabla_W R(W) \tag{8}$$

where $\gamma \in (0, 1)$ is the learning rate, W^t denotes the weight parameter based on GraphSAGE model at step t and $R(W)$ is the cumulative reward of the parameter.

The implementation of the GraphSAGE-based feature extraction algorithm is shown in Algorithm 2.

Algorithm 2. GraphSAGE-based feature extraction algorithm

Input: Undirected graph G, Node feature vectors $\{x_i, \forall v_i \in V\}$, sample size q, network depth L, weighting matrix W
Output: hidden feature vector of the node $\{h_i, \forall v_i \in V\}$

1: Initialize the initial hidden state of each node of the network $h_i^0 \leftarrow x_i, \forall v_i \in V$;
2: For each $J = 1, 2, \cdots L$ execute Equation (7);
3: For all $v_i \in V$ execute Equation (7);
4: Sample all neighboring nodes v_j of v_i with a total of q neighboring nodes to obtain the set of neighboring nodes $N_i, \forall v_i \in N_i$;
5: Extract neighborhood aggregation features of target nodes using aggregation function as follows: $h_{N_i}^J \leftarrow AGGREGATE\left(\left\{h_j^{J-1}, \forall v_i \in N_i\right\}\right)$;
6: Neighborhood aggregated features are concatenated with node features to generate hidden feature vectors using an aggregation function as follows: $h_i^J \leftarrow \sigma\left(W \cdot MEAN\left(\left\{h_j^{J-1}\right\} \cup \left\{h_{N_i}^J\right\}\right)\right)$
7: Paradigm normalization of results $h_i^J \leftarrow h_i^J / \left\|h_i^J\right\|_2$ is performed;
8: Obtain the final hidden feature vector of the node $h_i \leftarrow h_i^L, \forall v_i \in V$.

4. Routing Policy for DQN

In order to adapt to dynamic networks with frequent topological changes, GNN-DRL algorithm employs a distributed routing decision mechanism to assign routing decisions to each node instead of pre-constructing routing paths. DQN is executed and distributed at each node, while the centralized training of DQN is implemented to improve stability. In this paper, the Q value of each neighboring node (i.e., an action) is computed in order. The state representation of the link associated with the node is taken as the input to the DQN, and the Q value of the neighbor node is taken as the output of the DQN. After obtaining the Q values of all the neighbor nodes, the neighbor node with the largest Q value is selected as the next-hop node to forward the packet.

In each time slot τ, the hidden representation representing the network output (i.e., the GpaphSage model, whose output data are the network-side state h_i^τ) will be connected with the user-side state s_d^τ. Then the network state vector $h^\tau = \left[h_i^\tau, s_d^\tau\right]$ will be used as the input to the DQN model. The structure of the neural network model for the DQN agent in the end-to-end transmission routing problem is shown in Figure 3. The model takes the network link quality matrix as the initial state and feeds it into the neural network. The hidden layer of the neural network consists of a two-layer fully connected network and an activation function. Finally, the model outputs Q values corresponding to multiple candidate paths for end-to-end transmission. According to the Q values, the candidate paths can be selected as end-to-end transmission paths by a greedy strategy.

Figure 4 shows the model of the next node selection framework based on DQN which consists of three components: environment, agent and reward. The DRL agent is operated by interacting with the network simulation environment and the learning process is implemented by DQN. During execution progress, the node that saves the packet fixes its state and selects an action to determine the next hop node using DQN. After performing an action, the node receives a reward from its environment. During training, the centralized trainer draws a small random sample from the experience pool and updates its parameters by minimizing the loss of the DQN. After the parameters are updated, the centralized trainer sends the updated parameters to each node. Upon receiving the updated parameters, each node updates the parameters of its DQN. Then, the old experiences are deleted, and the new policy parameters are used to collect new experiences. This progress is repeated until convergence or a predefined criterion is reached.

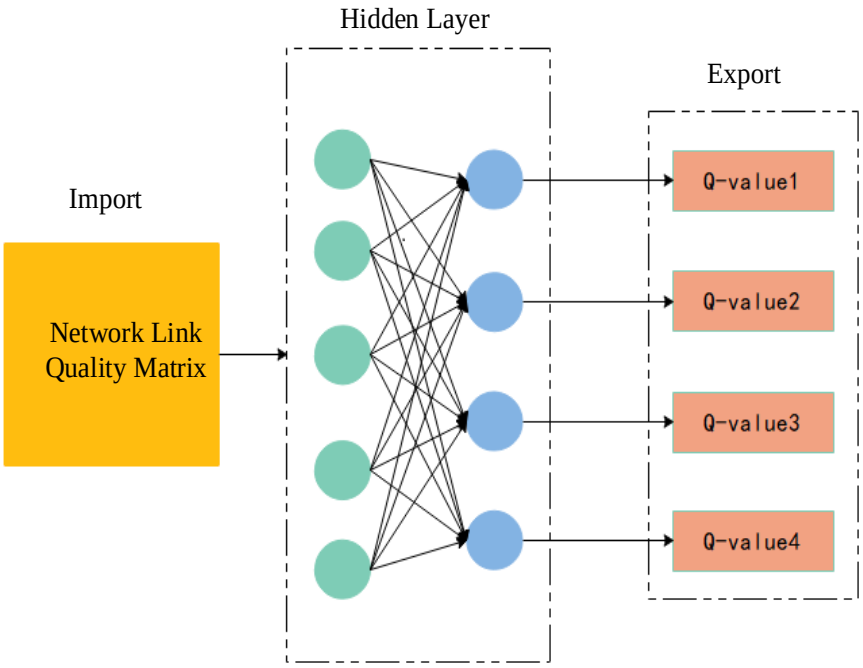

Figure 3. Architecture of the DQN agent.

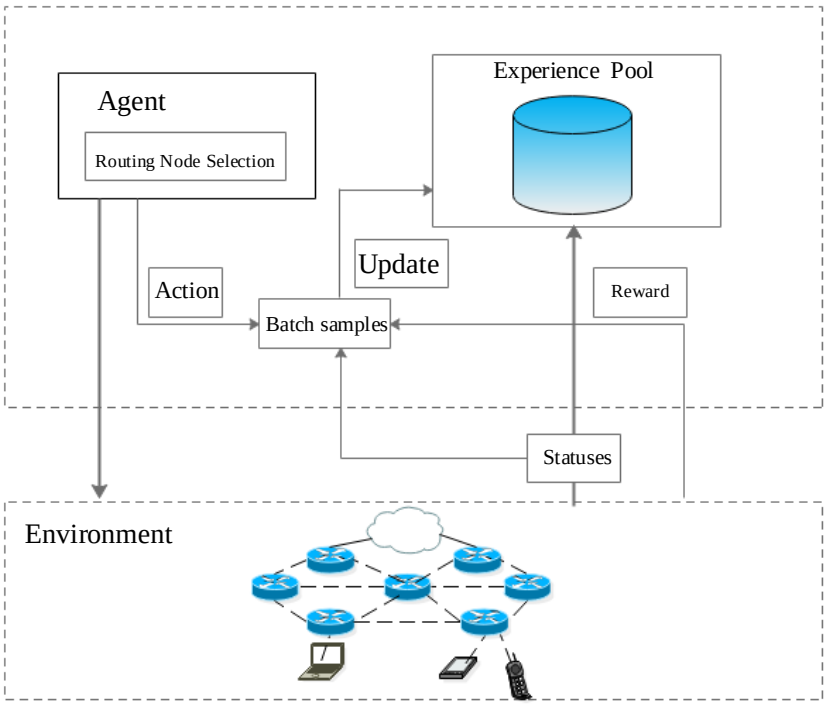

Figure 4. DQN-based next node selection framework.

At the beginning, the DQN agent goes through all traffic demands and tries to change its current routing configuration. For each traffic demand, the DQN agent evaluates and assigns the demand for each possible node on this traffic routing path with the GNN. The DQN outputs a Q value for each action and selects the action to be performed by the agent from the Q values of all actions. Finally, the network performs the selected action, which achieves a new state, a reward and a flag when the agent has completed iterating over all traffic demands. The reward is set to the minimum end-to-end delay between two steps.

Since the deviation of $Q^{\pi}(s,a)$ in the MDP solution of Equation (4) is usually very high, the dominance function $A^{\pi}(s,a)$ is proposed to replace the value function $Q^{\pi}(s,a)$, which is expressed as follows:

$$A^{\pi}(s_{\tau},a_{\tau}) = Q^{\pi}(s_{\tau},a_{\tau}) - V^{\pi}(s_{\tau}) = R_{\tau} - V^{\pi}(s_{\tau}) \tag{9}$$

where $V^{\pi}(s)$ is a function of the state value expected to be returned in state s. The superiority of $A^{\pi}(s,a)$ in selecting an action a from the set of actions in a particular state s can be seen. The policy π is represented by the neural network and its parameter θ, i.e., $\pi_{\theta}(a|s;\theta)$. And the policy parameters are updated using the parameter gradient as shown in Equation (10):

$$\nabla_{\theta} J(\theta) = E_{\pi_{\theta}}\left[\sum_{\tau=0}^{\infty} \nabla_{\theta} \log \pi_{\theta}(a_{\tau}|s_{\tau}) A^{\pi_{\theta}}(s_{\tau},a_{\tau})\right] \tag{10}$$

Specifically, the parameter θ is updated according to the current strategy and the dominance function $A^{\pi}(s,a)$.

In the distributed routing model, each agent updates the policy parameters using the above formulas, and the specific process of realizing the GNN-DRL algorithm is shown in Algorithm 3.

Algorithm 3. Process of Distributed Routing Algorithm for DQN Agents Based on GraphSAGE.

Input : traffic demand (src, dst, bw) flowing through agent i
Output: find the best next hop node for the route

1: Initialize the network G, neural network parameters θ and experience Pool D;
2: If $(i \neq d)$;
3: Feature extraction is performed based on GraphSAGE to obtain the final hidden feature vector of the node $h_i \leftarrow h_i^L, \forall v_i \in V$;
4: Combine user states to form a state space $s_i \leftarrow [h_i \Big|\Big| h_{(s,d,bw)}]$;
5: Select action a from $Q(s,a)$ using the strategy function;
6: Perform executable action a, obtain reward r and new status s';
7: Add the sample (s,a,r,s') to the experience pool D;
8: A random batch sample is drawn from the empirical pool. And the dominance function is computed according to Equation (9) so that the standard deviation function (loss function) is minimized;
9: Use the parameter gradient according to Equation (10) to update the policy parameter θ;
10: Reset the target action value function $\hat{Q} = Q$;
11: Save the model.

5. Performance Evaluation

Since the LEO networks lack of the dataset for training, we use NSFNet dataset instead of the LEO dataset in this section for training. Both LEO networks and NSFNet are hierarchical routing networks. And some characteristics of LEO networks are similar to those of NSFNet. Changing the connecting state of some links in NSFNet can monitor the topological changes in LEO networks. Usually the topological changes in LEO are caused by link failure and node failure, and in this paper, they can be monitored by disabling the nodes in the NSFNet, which will cause link failure and node failure. Other than that, the node and link state are almost the same. The network modeling open-source dataset KDN (knowledge-defined networking) is used in this chapter, which is based on the real environment generated through OMNet++. The NSFNet network topology in KDN is shown in Figure 5. The NSFNet network contains the following: network node distribution, node attribute characteristics, traffic matrix, transmission delay, link bandwidth, etc.

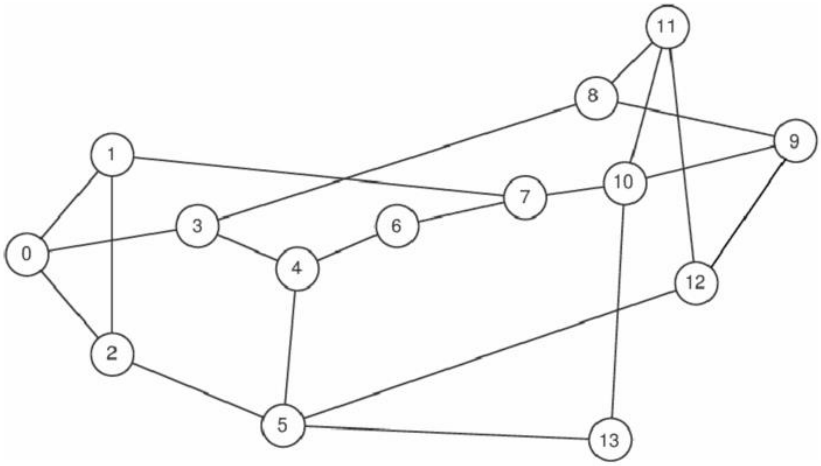

Figure 5. NSFNet network topology.

The GNN-DRL modeling algorithm uses the KDN dataset and is implemented based on the TensorFlow and OpenAIGym frameworks. All experiments were performed using standard hardware (Ubuntu 20.04.1 LTS (Canonical and friends, Boston, MA, USA), AMD Riptide 9 3950x16-core processor (AMD, Santa Clara, CA, USA), GTX-3080 graphics card (NVIDIA, Santa Clara, CA, USA)) without using any special hardware accelerators. The simulation environment is configured by initializing the link characteristics of each topology with its respective link capacities. In order to make the weights for all links equal, the network is set to have a bandwidth capacity of 100 Mbps for each link. Traffic demand is initially distributed on each node according to precomputed routing paths, then the simulation environment is sequentially updated when training by traffic demand.

The DRL agent uses GraphSAGE to extract the hidden feature state of the links. In each execution of the GNN, T = 5 message passing steps are run, and a small batch of 50 samples is used. The optimizer used for training is the Adam optimizer, which has an initial learning rate of 2×10^{-4} and follows an exponential learning rate decay during training.

During the training of the GNN-DRL algorithm, the reward function converges very quickly. In order to evaluate the effectiveness of the routing algorithms based on graph neural networks, we will give the reward curves for all machine-learning-based algorithms in our experiments during online learning. For simplicity, the reward curves will be normalized as follows:

$$R_{normalized} = (R - R_{min})(R_{max} - R_{min}). \tag{11}$$

where R_{min} and R_{max} are the minimum and maximum rewards during online learning. The normalized reward curve is shown in Figure 6, and the results show that the cumulative reward of the GNN-DRL algorithm is better than that of the DQN algorithm.

Three routing algorithms are compared in the simulation implementation: Dijkstra, DQN and GNN-DRL. The GNN-DRL is the proposed graph neural-network-based routing optimization method for LEO satellite networks in this chapter. After a comparison is made, the GNN-DRL algorithm can find a feasible solution to significantly improve the average network throughput and reduce the average end-to-end delay under different traffic demands, as shown in Figures 7 and 8. It can be seen that the GNN-DRL algorithm has better performance than that of the other two methods in terms of average throughput and end-to-end delay. Compared with Dijkstra and DQN, the GNN-DRL algorithm increases the average throughput by 29.47% and 18.42%, respectively, and reduces the average end-to-end delay by 39.76% and 15.29%, respectively.

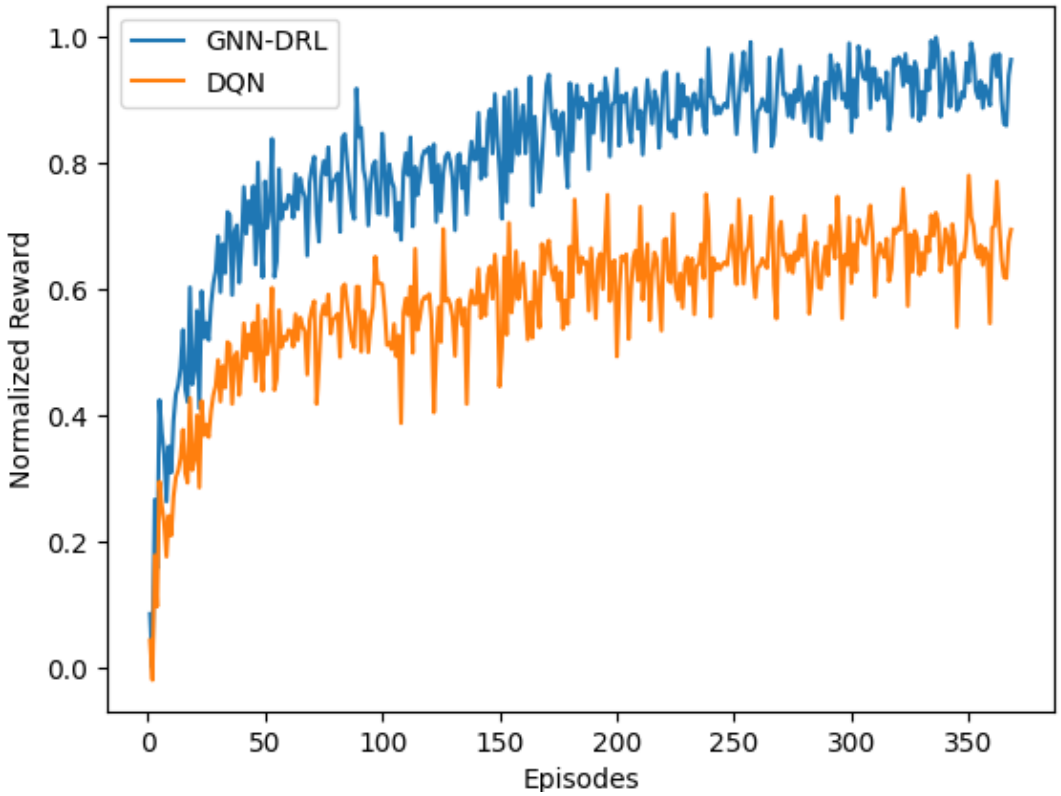

Figure 6. Training process of GNN-DRL algorithm.

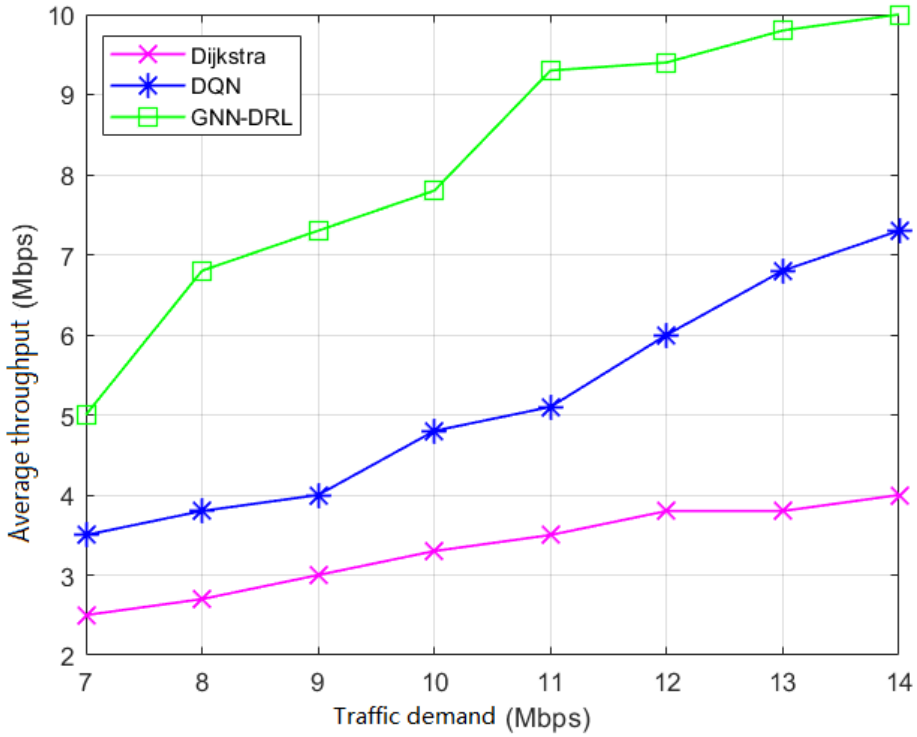

Figure 7. Average throughput for different traffic demands.

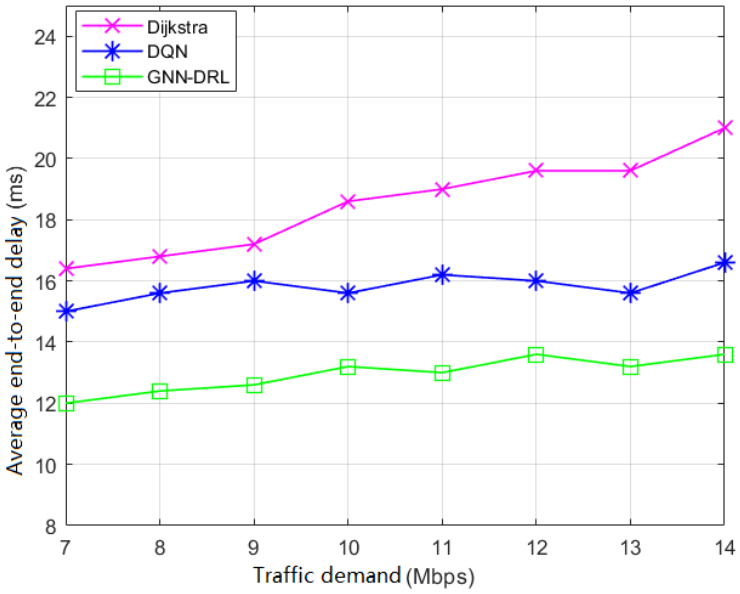

Figure 8. Average end-to-end delay for different traffic demands.

In order to evaluate the performances of Dijkstra, DQN and GNN-DRL algorithms with changes in the network topology, we consider a case in which there are node failures in the network. In this test, the NSFNet topology is disconnected with different numbers of selected nodes to simulate the network node failures in a real network operational environment. And the experimental results are shown in Figure 9. The GNN-DRL method proposed in this paper outperforms Dijkstra and DQN on topologies not seen in the training phase. It can be seen that GNN-DRL has better performance than that of the other two methods in coping with changes in the network topology and always has lower end-to-end delay than that of Dijkstra and DQN. The average end-to-end delay of GNN-DRL is reduced by 29.46% and 17.29% compared to Dijkstra and DQN, respectively. This proves that the GNN-DRL algorithm proposed in this paper can adapt to different network scenarios and can operate in highly dynamic networks (e.g., traffic variations, link failures and node failures), even if the network topology changes frequently in such a scenario.

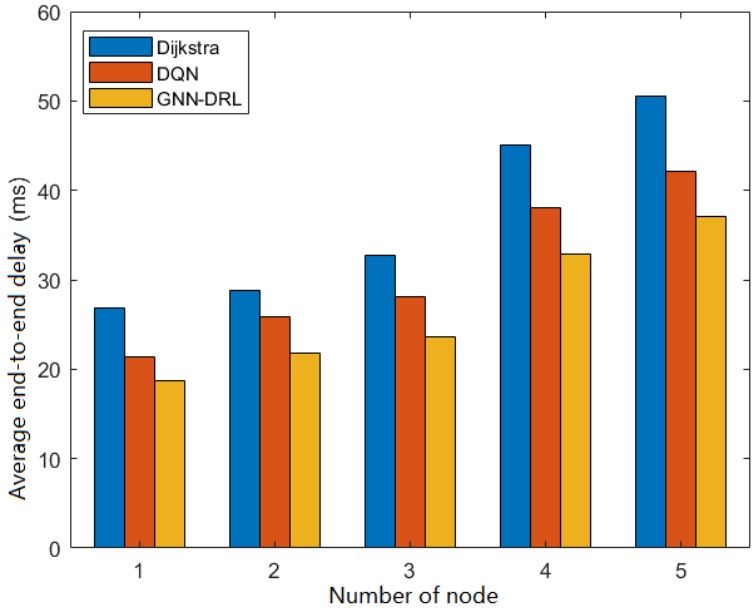

Figure 9. Average end-to-end delay for network topological change scenarios.

Due to the small number of nodes in the dataset used for training, it needs to be considered whether the algorithm is applicable to large networks. After completing the training with different routing schemes under NSFNet topology, the trained routing algorithm was applied to networks with different number of nodes to test the changes in the average end-to-end delay and average throughput of the network as the number of network nodes increases.

Figure 10 shows the simulation results of the average end-to-end delay of the network with the variation in the number of nodes. As the number of nodes in the network keeps increasing, the average end-to-end delay of the three algorithms keeps becoming larger. But it can be seen that the average end-to-end delay of the GNN-DRL algorithm proposed in this paper is always lower than that of the other two routing algorithms. At thirty nodes, the average end-to-end delay of the GNN-DRL algorithm is 30% and 22% lower than those of the Dijksta and DQN algorithms, respectively. Figure 11 shows the simulation results of the average throughput of the network with the variation in the number of nodes. As the number of nodes in the network increases, the average throughput of the three algorithms becomes smaller, but it can be seen that the average throughput of the GNN-DRL algorithm is overall higher than that of the other two routing algorithms.

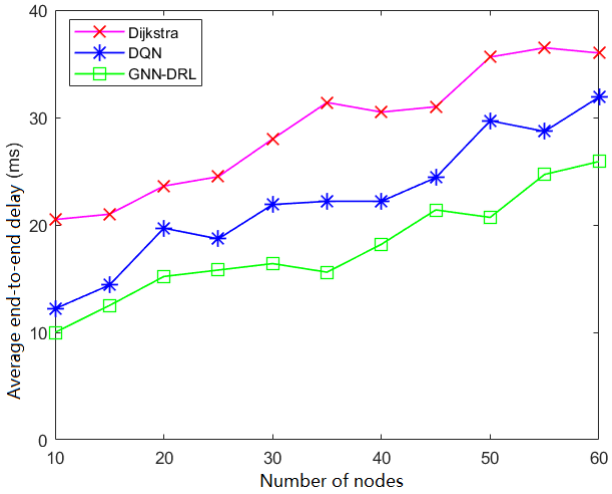

Figure 10. Average end-to-end delay of the network with different numbers of nodes.

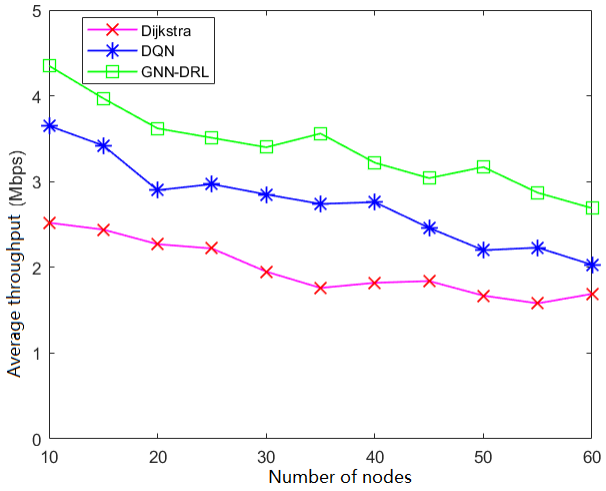

Figure 11. Average network throughput for different numbers of nodes.

6. Conclusions

This paper implements a routing optimization algorithm for low-orbit satellite networks based on graph neural networks. A graph neural-network-based algorithm for

low-orbit satellite networks is designed to adapt to dynamic networks with frequent topological changes and to solve the network congestion problem. The GNN-DRL algorithm proposed in this paper optimizes network routing by modeling the graph data in the network with a GNN feature engineering model, generating hidden feature vector representations of its own nodes with graph neural networks and then using a fully distributed multi-intelligence DRL routing model to make routing decisions with a deep reinforcement learning algorithm. Compared with Dijkstra's algorithm as well as the traditional DQN algorithm, the GNN-DRL algorithm proposed in this paper not only improves the overall network throughput but also reduces the average end-to-end delay. Compared with the Dijkstra and DQN algorithms, the average throughput of the GNN-DRL algorithm increases by 29.47% and 18.42%, respectively, and the average end-to-end delay is reduced by 39.76% and 15.29%, respectively. In addition, the model used in the GNN-DRL approach is able to cope with network topological changes (e.g., traffic variations, link failures and node failures) and is more suitable for real networks.

Author Contributions: Writing—original draft, Y.S.; Writing—review & editing, W.W., X.Z. and H.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Natural Science Foundation of China (Grant Nos. 92367102, 92067101) and the Key Research and Development Plan of Jiangsu Province (Grant No. BE2021013-3).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

Terms	Abbreviation
Low Earth orbit	LEO
Graph sample and aggregate	GraphSAGE
Deep reinforcement learning	DRL
Deep Q-Network	DQN
National Science Foundation Network	NSFNet
Knowledge-defined networking	KDN

References

1. Wang, H.; Ran, Y.; Zhao, L.; Wang, J.; Luo, J.; Zhang, T. GRouting: Dynamic Routing for LEO Satellite Networks with Graph-based Deep Reinforcement Learning. In Proceedings of the 2021 4th International Conference on Hot Information-Centric Networking (HotICN), Nanjing, China, 25–27 November 2021; pp. 123–128. [CrossRef]
2. Zuo, P.; Wang, C.; Yao, Z.; Hou, S.; Jiang, H. An Intelligent Routing Algorithm for LEO Satellites Based on Deep Reinforcement Learning. In Proceedings of the 2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall), Norman, OK, USA, 27–30 September 2021; pp. 1–5. [CrossRef]
3. Cai, J.; Wang, C.; Lei, M.; Zhao, M.-J. An Intelligent Routing Algorithm Based on Prioritized Replay Double DQN for MANET. In Proceedings of the 2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall), Victoria, BC, Canada, 18 November–16 December 2020; pp. 1–5. [CrossRef]
4. Xu, X.; Lu, Y.; Fu, Q. Applying Graph Neural Network in Deep Reinforcement Learning to Optimize Wireless Network Routing. In Proceedings of the 2021 Ninth International Conference on Advanced Cloud and Big Data (CBD), Xi'an, China, 26–27 March 2022; pp. 218–223. [CrossRef]
5. Nguyen, T.-V.; Tran, T.-N.; An, B. A Deep Q-Learning Design for Energy Harvesting QoS Routing in IoT-enabled Cognitive MANETs. In Proceedings of the 2021 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Jeju Island, Republic of Korea, 13–16 April 2021; pp. 401–406. [CrossRef]
6. Wang, Z.; Han, R.; Li, H.; Knoblock, E.J.; Apaza, R.D.; Gasper, M.R. Deep Reinforcement Learning based Routing in an Air-to-Air Ad-hoc Network. In Proceedings of the 2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC), Portsmouth, VA, USA, 18–22 September 2022; pp. 1–6. [CrossRef]

7. Tang, X.; Xu, Y.; Pan, S. Intelligent Routing Algorithm Based on Graph Convolutional Neural Networks. *Comput. Eng.* **2022**, *48*, 38–45.
8. Wang, X.; Wang, S.; Liang, X.; Zhao, D.; Huang, J.; Xu, X.; Dai, B.; Miao, Q. Deep Reinforcement Learning: A Survey. *IEEE Trans. Neural Netw. Learn. Syst.* **2024**, *35*, 5064–5078. [CrossRef] [PubMed]
9. Liu, X.; Zhang, H.; Long, K.; Nallanathan, A.; Leung, V.C.M. Deep Dyna-Reinforcement Learning Based on Random Access Control in LEO Satellite IoT Networks. *IEEE Internet Things J.* **2022**, *9*, 14818–14828. [CrossRef]
10. Zhang, Z.; Cui, P.; Zhu, W. Deep Learning on Graphs: A Survey. *IEEE Trans. Knowl. Data Eng.* **2022**, *34*, 249–270. [CrossRef]
11. Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; Yu, P.S. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *32*, 4–24. [CrossRef] [PubMed]
12. Oloulade, B.M.; Gao, J.; Chen, J.; Lyu, T.; Al-Sabri, R. Graph neural architecture search: A survey. *Tsinghua Sci. Technol.* **2022**, *27*, 692–708. [CrossRef]
13. Hamilton, W.; Ying, Z.T.; Leskovec, J. Inductive Representation Learning on Large Graphs. In Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017.
14. Baeza, V.M.; Ortiz, F.; Lagunas, E.; Abdu, T.S.; Chatzinotas, S. Multi-Criteria Ground Segment Dimensioning for Non-Geostationary Satellite Constellations. In Proceedings of the 2023 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), Gothenburg, Sweden, 6–9 June 2023. [CrossRef]