Hindawi
Wireless Communications and Mobile Computing
Volume 2023, Article ID 6593938, 16 pages
https://doi.org/10.1155/2023/6593938

WILEY | Hindawi

Research Article

An Enhanced Deep Reinforcement Learning-Based Global Router for VLSI Design

Saijuan Xu[1], **Liliang Yang**[2,3] **and Genggeng Liu**[2,3]

[1]*Department of Information Engineering, Fujian Business University, Fuzhou, China*
[2]*College of Computer and Data Science, Fuzhou University, Fuzhou, China*
[3]*Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou University, Fuzhou, China*

Correspondence should be addressed to Genggeng Liu; liu_genggeng@126.com

Received 11 November 2022; Revised 23 February 2023; Accepted 25 April 2023; Published 5 May 2023

Academic Editor: Kaize Shi

Global routing is a crucial step in the design of Very Large-Scale Integration (VLSI) circuits. However, most of the existing methods are heuristic algorithms, which cannot conjointly optimize the subproblems of global routing, resulting in congestion and overflow. In response to this challenge, an enhanced Deep Reinforcement Learning- (DRL-) based global router has been proposed, which comprises the following effective strategies. First, to avoid the overestimation problem generated by *Q*-learning, the proposed global router adopts the Double Deep *Q*-Network (DDQN) model. The DDQN-based global router has better performance in wire length optimization and convergence. Second, to avoid the agent from learning redundant information, an action elimination method is added to the action selection part, which significantly enhances the convergence performance of the training process. Third, to avoid the unfair allocation problem of routing resources in serial training, concurrent training is proposed to enhance the routability. Fourth, to reduce wire length and disperse routing resources, a new reward function is proposed to guide the agent to learn better routing solutions regarding wire length and congestion standard deviation. Experimental results demonstrate that the proposed algorithm outperforms others in several important performance metrics, including wire length, convergence performance, routability, and congestion standard deviation. In conclusion, the proposed enhanced DRL-based global router is a promising approach for solving the global routing problem in VLSI design, which can achieve superior performance compared to the heuristic method and DRL-based global router.

1. Introduction

In Very Large-Scale Integration (VLSI), millions of pins are integrated into a chip. A set of pins with the same potential forms a net that must be connected by wires [1]. With the advancement of the VLSI manufacturing process, the density of nets on a chip and the required wire length are increasing. Two-pin net routing is the simplest global routing problem, but it still belongs to the NP-hard problem [2], and the optimal solution cannot be computed in polynomial time. Therefore, one of the challenges in the physical design phase is to get a feasible routing solution in a reasonable time while minimizing the wire length.

The VLSI routing phase is divided into global routing [3] and detailed routing [4]. The routing area of global routing is divided into uniformly sized grids, and the connections between pins are abstracted as intergrid connections [5]. This phase requires a reasonable allocation of wires to the grids and guides detailed routing. The metrics used to evaluate the quality of global routing include overflow, wire length, and congestion [6, 7]. The overflow determines the yield of the chip. Therefore, this paper takes the overflow as the main design goal and optimizes the wire length and congestion based on the overflow as 0.

Reinforcement learning (RL) is different from supervised and unsupervised learning [8] which belongs to the third type of machine learning. In RL, an agent interacts with its environment by trial. A reward is returned to the agent by its environment when the agent takes action [9]. The goal of RL is to maximize the cumulative rewards obtained by the agent. However, RL often faces the problem of the excessive number of states when dealing with high-dimensional spaces. With the development of deep learning, the Deep Reinforcement Learning (DRL) algorithm is developed by combining artificial neural networks with RL [10], which makes it possible for RL to solve the policy decision in a high-dimensional space [11].

Global routing problems can be divided into several two-pin path planning subproblems [12]. Path planning is a sequential decision process with Markov property [13]. RL can solve this type of problem well. In path planning, an agent performs discrete actions for random decision-making and receives rewards in a finite number of states. The future state of the agent only depends on the current state, and all states in the environment are observable, which is called the Markov Decision Process (MDP).

This paper is aimed at improving the design of a global router by utilizing the conjoint optimization ability of DRL, which outperforms heuristic methods in terms of overflow and wire length [14]. In addition, this research seeks to surpass existing DRL-based global routers in terms of wire length, routability, and convergence performance. To achieve this goal, an enhanced DRL-based global router is designed, and four effective optimization strategies are proposed. The main contributions of this paper are as follows:

(i) First, to avoid the overestimation problem generated by Q-learning, a global router based on Double Deep Q-Network (DDQN) [15] is proposed. Experimental results demonstrate that the DDQN-based global router outperforms the Deep Q-Network (DQN) based on wire length and convergence performance

(ii) Second, to reduce the redundant information and enhance the convergence performance of the model, we propose an action reduction method, which is proved to have an enhancement on the convergence performance of the model through experimental results

(iii) Third, since the earlier nets occupy the routing resources of the later nets, resulting in unfair resource allocation, we propose a concurrent training method to solve the unfair resource allocation problem as much as possible

(iv) Fourth, a new reward function is designed to encourage wire sharing and decentralize routing resources reasonably. The new reward function has good optimization results in wire length and congestion performance compared with the reward function in [16]. In summary, the experimental results show that this algorithm achieves better results in several important metrics

2. Related Work

Overflow and wire length are the two main optimization goals of the global routing problem, as well as the main optimization goal of the 2007 and 2008 ISPD competitions.

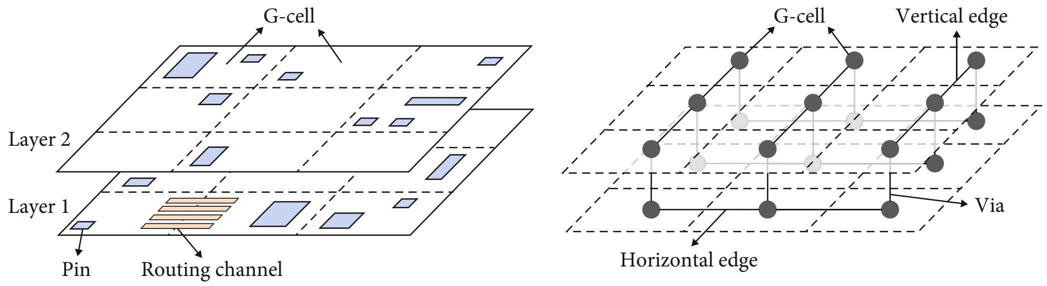
这也太老了

The method to solve global routing is divided into serial routing and parallel routing [17]. Serial routing usually sorts nets in a specific order and routes them one by one; this method is fast. However, there is an unfair phenomenon: the routing difficulty of the earlier nets has sufficient routing resources (meaning that the capacity of each edge in the routing area is large), while most of the later nets have tight routing resources, so the serial routing method usually rips up part of the nets and reroutes them. The existing methods based on serial routing are as follows: [18] proposed a dynamic routing mode, and this mode is driven by a movable wires method, which quickly and accurately provides a routing solution driven by routability. [19] proposed enhanced variants of extreme edge shift and edge contraction to achieve rapid exploration of candidate paths. [20] proposed a circular fixed-order monotonic routing and a high-performance congestion-driven 2D global router.

The parallel method routes multiple nets at the same time [21], solving the unfairness of routing resources in a serial method, but it is often very time-consuming and even impossible to solve, mainly based on the commodity flow model [22] and integer linear programming model [23].

Heuristic methods are effective in global routing. [24] presented a rectilinear SMT (RSMT) algorithm based on discrete particle swarm optimization (DPSO) to optimize wire length efficiently. [25] used the DPSO algorithm and the firefly algorithm to construct the global routing solutions. [26] presented a DPSO-based multilayer obstacle avoidance X-architecture SMT (XSMT), which uses an effective penalty mechanism to help particles avoid obstacles. [27] introduced five commonly used swarm intelligence technologies and related models and used them in three classic routing problems: SMT, global routing, and detail routing. [28] proposed a new type of DPSO and multistage transformation to be used to construct RSMT and XSMT. The simulation results of industrial circuits show that this method can obtain high-quality routing solutions. [29] presented an XSMT construction method based on multistrategy optimization Discrete Differential Evolution (DDE), which significantly reduces XSMT wire length. [30] proposed an XSMT algorithm based on social learning DPSO and an effective two-stage construction method to achieve the best wire length optimization effect. [31] presented an efficient VLSI routing algorithm employing novel DPSO and multistage transformation to build XSMT. [32] presented a unified algorithm for XSMT and RSMT construction based on a hybrid transformation strategy (HTS) and self-adapting DPSO. [33] presented an effective DPSO-based power-driven length-restricted X-routing algorithm. The algorithm can achieve the best wire length cost at a very fast speed under the constraint of restricted wire length.

DRL-based global routing is different from heuristic-based global routing. Compared with heuristic methods, DRL has more flexibility and conjoint optimization capabilities. [16] proposed the DRL-based global routing algorithm for the first time. The results of [16] show that its overall performance is better than the sequential A^* algorithm. To this end, this paper proposes four effective optimization strategies for the DRL algorithm model, to further enhance



(a) Global routing structure

(b) Grid graph

Figure 1: Global routing structure and its corresponding grid graph.

the performance of the DRL-based global router. We compare the enhanced DRL-based global router with [16] and the sequential A^* in Section 5.

3. Preliminaries

3.1. Global Routing. In VLSI global routing, the routing area is a multilayer structure. Each layer is divided into several rectangles of the same size, and each rectangle is called a G-cell. Figure 1(a) shows a two-layer global routing structure, where the white rectangular areas are G-cells, the blue rectangles are pins, there are several pins in a single G-cell, and the yellow rectangles are the routing channel. The number of routing channels determines the maximum number of wires that can be accommodated. Usually, only the wires between G-cells are considered in global routing, and the wires inside the G-cell are ignored, so the model in Figure 1(a) can be further simplified. Figure 1(b) shows the simplified structure. Each G-cell is abstracted as a dot, and the routing channels between two G-cells are abstracted as a black line, called an edge. Due to the limitations of the manufacturing process, a single-layer routing area can only have unidirectional edges. In Figure 1(b), there are only horizontal edges between the G-cells of the first layer and only vertical edges of the second layer. When routing a circuit, it is often necessary for a global router to change direction to reach another layer. In order to accomplish this, the global router must pass through a via, which is a vertical connection between layers in a multilayered printed circuit board or integrated circuit.

In a given set of pins, a subset of pins with the same potential is a net, and the pins of the same net need to be connected. The maximum number of wires that an edge can accommodate is called capacity. If the capacity of each edge is e_c and the number of wires passing through the edge is e_d, the congestion of the edge is $e_{cg} = e_d/e_c$. When e_d is greater than e_c, overflow occurs, and the overflow number is $e_{of} = \max(0, e_d - e_c)$. Overflow will affect the yield of the chip. Therefore, the primary routing goal of this algorithm is to guarantee the number of overflows is 0 and minimize the wire length WL and the standard deviation Std of congestion.

3.2. Deep Reinforcement Learning. RL consists of five core elements: state s_t, action a_t, reward r_t, policy π, and action-value $Q(s,a)$, where t is the current time. s_t is the current state of the environment and the agent. a_t is the action taken by the agent at time t. r_t is the reward obtained by the agent taking the action a_t from state s_t to s_{t+1}. π is the basis for the agent to take action a_t, which is usually a conditional probability distribution, that is, the probability of taking action a_t under the condition of the state s_t, as shown in Formula (1). $Q(s,a)$ is an expectation function, which is the expectation of the subsequent delayed rewards obtained by the agent after taking action a_t in state s_t and policy π, as shown in Formula (2):

$$\pi(a_t|s_t) = P(a = a_t|s = s_t), \quad (1)$$

s下选a的概率

$$Q(s,a) = E_\pi\left(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s_t = s, a_t = a\right). \quad (2)$$

包括以后的总和

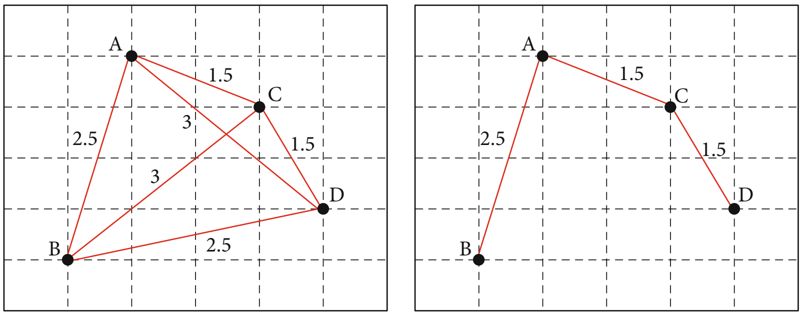
The goal of RL is to search for a policy that maximizes the cumulative rewards, and the predictive action-value function $Q(s,a)$ is often different from the target action-value function $Q^*(s,a)$. So the main process of RL is to continuously try and modify the predictive action-value so as to make the predictive action-value function fit the target action-value function, as shown in

$$Q^*(s,a) = \max_\pi E(r_{t+1}|s_t = s, a_t = a, \pi), \quad (3)$$

where γ is a discount factor, $\gamma \in [0,1]$. If $\gamma = 0$, the current action-value is only determined by the current reward. If $\gamma = 1$, the subsequently delayed rewards have the same weight as the current reward. Generally, a decimal between 0 and 1 is taken so that the weight of the current reward is higher than subsequent delay rewards. The action-value function can be derived into the Bellman equation form, as shown in

$$Q(s,a) = E_\pi\left(r_t + \gamma \max_{a_t} Q^*(s_t, a_t) | s_t = s, a_t = a\right). \quad (4)$$

DRL introduces ω to approximate the action-value function. There are many types of approximation methods, and the most commonly used is the neural network method. The neural network in the DQN algorithm is called the Q-network, and DQN uses two Q-networks: they are the predictive Q-network and the target Q-network. The predictive Q-network is used to predict the action-value corresponding to the current state and choose



(a) Complete graph of multipin net

(b) Two-pin net obtained by decomposition

Figure 2: Decomposition of a multipin net.

action according to the ε-greedy policy, as shown in Formula (5). The target Q-network is used to calculate the target Q-value using a delayed update method. The parameters of the predictive Q-network are copied to the target Q-network at regular intervals. The DQN algorithm uses the experience replay buffer to store the experience $<s, a, r, s', \text{end}>$ generated by the interaction between the agent and the environment, where s' represents the next state agent reached after taking action a in state s, and end is a binary variable that represents whether the state s is in the terminated state or not:

$$a(s) = \begin{cases} \operatorname{argmax}_a Q(s, a; \omega), & \text{rnd} < \varepsilon, \\ \text{random}(a \in A), & \text{else}, \end{cases} \tag{5}$$

where $\varepsilon, \text{rnd} \in [0, 1]$, rnd is a random number between 0 and 1. When $\text{rnd} < \varepsilon$, the action with maximum action-value is selected greedily; otherwise, a random action is selected.

The training samples are randomly sampled from the experience replay buffer. The states of the samples are the input of the target Q-network. The action-values of the actions are the output of the target Q-network. DQN is always selected as the maximum action-value to calculate the target Q-value, as shown in

$$y_i = r_i + \gamma \max_a Q^*\left(s_i, a_i; \omega'\right), \tag{6}$$

where ω' represents the parameters of the target Q-network, copied from ω at regular intervals. $Q^*(s, a)$ is the target action-value function.

4. DDQN-Based Global Router

4.1. Decomposition of Multipin Nets. In global routing, a net usually contains two or more pins. If a net has more than two pins, we call it a multipin net. It is difficult to directly route a multipin net. Before routing starts, it is usually necessary to decompose the multipin net into two-pin nets. To shorten the wire length of two-pin nets after decomposition, the Prim algorithm is used to construct a minimum spanning tree (MST) of a net. All pins are mapped on a two-dimensional plane, and the weight between each two-pin net is the Manhattan distance.

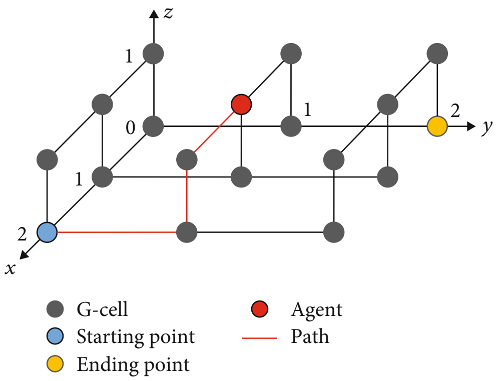


Figure 3: An example of global routing.

Figure 2 shows an example of decomposing a multipin net. Figure 2(a) shows a complete graph of a net with four pins. Pin A is selected as the starting point to construct an MST. Figure 2(b) shows the MST structure of Figure 2(a). Based on the tree structure, a four-pin net is decomposed into 3 two-pin nets (AB, AC, and CD).

The longer the distance between two pins, the more difficult it is to find a path between them. Therefore, the DRL training process proceeds in descending order of the two-pin distance.

这个有用

4.2. Encoding. A DDQN-based global router is executed in a three-dimensional environment, and three-dimensional coordinates represent the position of the pin. A two-pin net can be determined by the coordinates of the pins at both ends. Choose either end of the two-pin net as the starting point and the other end as the ending point. Leading out at the starting point, the current routing position of the wires (that is, the agent's position) can be represented by the coordinates of the nonstarting end of the wires, and the edge capacities of the agent position also need to be observed. Therefore, a 15-bit code is used; the starting point, the ending point, and the agent's position are all represented by a 3-bit code; and a 6-bit code represents the edge capacities in six directions.

In order to describe the encoding method more clearly, Figure 3 shows a coding example, the starting point coordinate is $(2, 0, 0)$, and the ending point coordinate is $(0, 2, 0)$. The agent's position (shown by the red dot) is (1, 1, 1), and the red wires are the path the agent traverses. Assuming that the capacity of each edge is 2, in the order of left, right, front, back, up, and down, the capacities of the six directions observed by the agent are 0, 0, 1, 2, 0, and 2. In summary, the current state can be encoded as 2 0 0 0 2 0 1 1 1 0 0 1 2 0 2.

The input of the Q-network is state codes, and the corresponding output is the action-value of 6 directions. However, in the routing example given in Figure 3, the number of actions the agent can perform is fewer than 6 at the coordinate (1, 1, 1). In fact, in the global routing problem of the two-layer structure, the maximum number of actions that the agent can perform is 3, and the remaining three actions are redundant. Thus, when an agent selects actions, the actions that cannot be performed are first eliminated to prevent the agent from performing redundant actions during training, storing redundant experience, and then learning redundant information. In Section 5, the reward line charts with and without the action elimination method are compared to verify its effectiveness.

4.3. Initialization of Experience Replay Buffer. Experience replay is an essential mechanism of DQN. It is used to replace the Q-table in Q-learning and update the parameters of the Q-network. A piece of five-tuple information obtained by the agent will be stored in the experience replay buffer every time it interacts with the environment. The storage principle of the experience replay buffer is first-in and first-out. When the experience replay buffer is full, the earliest stored experience is deleted first.

There are usually two methods to initialize the experience replay buffer. The first method lets the agent explore randomly in the environment, stores the acquired experience, and then starts training the Q-network when the experience replay buffer is full. The second method uses a heuristic algorithm to search for the path in advance and burn it into the experience replay buffer.

We use the second method to initialize the experience replay buffer. The A^*-based global router is used to obtain the initial routing results, where $f(n) = g(n) + h(n)$ is estimated function. Cost function $g(n)$ represents the cost of performing step n. If overflow occurs, $g(n) = -1000$. In other cases, $g(n) = -1$. Heuristic function $h(n)$ is used to estimate the cost of reaching the target position in the current position, which is expressed by the Manhattan distance from the current position to the target position.

Figure 4 shows a reward line chart of the two initialization methods. The blue line is the reward value of the first initialization method, and the red line is the reward value of the A^* initialization method. The test example adopts benchmark 1 of Section 5. It can be seen that the convergence speed of the red line is much higher than that of the blue line. The A^* initialization method is used to make the model converge to a better solution in fewer episodes than the first method.

4.4. DDQN Implement. From the calculation formula (Formula (6)) of the target Q-value of DQN, it can be seen that DQN always selects the maximum action-value to calculate the target Q-value. Although greedily selecting the maximum action-value can make the algorithm get closer to the estimated target Q-value quickly, the calculated model may have a deviation compared with the actual result, causing the problem of overestimation. In this section, we will propose a DDQN-based global router to solve this problem.

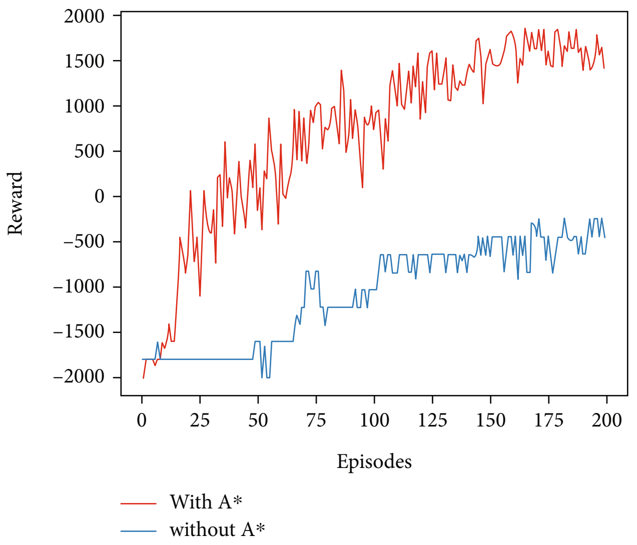


Figure 4: A reward line chart of two initialization methods.

In DDQN, the selection of actions is carried out in the predictive Q-network, and the evaluation of the actions is carried out in the target Q-network. DDQN replaces the action selection method with the maximum Q-value in the target Q-network to the predictive Q-network, as shown in Formula (7). The calculation formula of the target Q-value is also modified accordingly, as shown in Formula (8):

$$a(s_i\,;\omega) = \mathrm{argmax}_{a'} Q\left(s_i, a'\,;\omega\right), \tag{7}$$

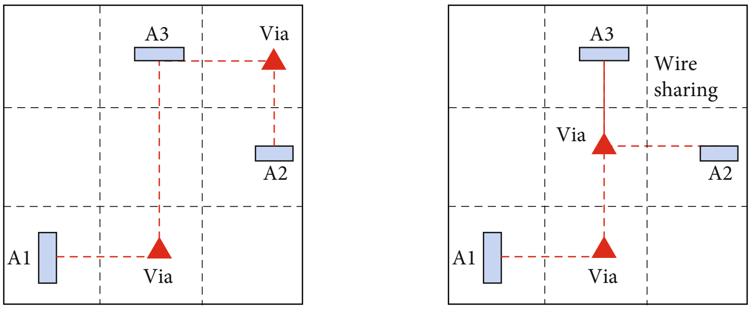
$$y_i = r_i + \gamma Q^*\left(s_i, a(s_i\,;\omega)\,;\omega'\right), \tag{8}$$

where argmax is the function that maximizes the independent variable. Compared with DQN, DDQN modifies the selection of actions when getting the target Q-value.

This paper uses the DDQN algorithm as the global router design framework. The performance comparison of DDQN and DQN in global routing will be discussed in detail in Section 5, and then, the detailed design part of the DDQN-based global router will be introduced as follows.

4.4.1. Reward Design. The environment will reward any action the agent takes. A reward value can be positive, negative, and zero. A negative value means that the environment imposes adverse penalties on the agent, and DDQN maximizes a cumulative reward. [16] considers shortening the wire length as the optimization goal and has the following reward function:

$$r(a, s_{i+1}) = \begin{cases} +100, & s_{i+1} \text{ is end,} \\ -1, & \text{else,} \end{cases} \tag{9}$$



(a) A global routing solution without wire sharing (b) A global routing solution with wire sharing

Figure 5: An example of wire sharing.

where the agent gets a reward value of +100 when it reaches the target state; in other cases, whether the agent successfully executes an action or is forced to stay in place because the edge capacity is 0, it will get a reward value of -1. This reward function enables the agent to reduce the actions performed in the environment as much as possible to reduce the wire length.

However, the above method does not reflect the actual routing situation and lacks consideration for sharing the net's wires. As shown in Figures 5(a) and 5(b), Pin A1, Pin A2, and Pin A3 belong to the same net. The two-pin A13 and two-pin A23 are obtained by the net decomposition method. Using the reward function of Formula (9) to calculate, the cumulative reward of Figures 5(a) and 5(b) is both 193. Figure 5(b) makes the wire length shorter than Figure 5(a) through the sharing of wires. Therefore, Formula (9) is unreasonable. We mark the paths that have been traversed in the net and set the reward value to 0 when wire sharing occurs. In addition, we also need to consider the congestion of the routing result. A large number of routing resources concentrated in a certain location will affect the performance of the chip. We adjust the reward value of the high and low congestion areas to obtain the following reward function:

$$r(a, s_{i+1}) = \begin{cases} +100, & s_{i+1} \text{ is end}, \\ 0, & s_{i+1} \text{ isn't end and sharing}, \\ (e_{\mathrm{d}} - e_{\mathrm{c}}/2)/e_{\mathrm{c}} - 1, & \text{else}, \end{cases} \tag{10}$$

where e_{c} and e_{d} are the capacity and demand of an edge, respectively. According to this reward function, we take half of e_{c} as the threshold, and the result of $e_{\mathrm{c}}/2$ is rounded down. If e_{d} is higher than $e_{\mathrm{c}}/2$, a reward $r < 0$ is given; otherwise, a reward $r \geq 0$ will be given. The experimental comparison of the two reward functions will be given in Section 5.

4.4.2. Neural Network Architecture. Both the evaluation Q-network and the target Q-network are three-layer fully connected neural networks. The number of neurons in the first to third layers is 32, 64, and 32, respectively. The activation function is the ReLU linear rectification function. The input layer size is 15, and the output layer size is 6.

4.4.3. Episode. Path planning for all two-pin nets is an episode. After an episode, the complete execution policy of the global routing will be obtained. DRL will perform repeated training under the same benchmark until the algorithm reaches a convergence state.

4.4.4. Max Step. During the training process, the agent may not find the target state for a long time or cannot find the target state at all, thus falling into an infinite loop. Therefore, each two-pin net needs to be assigned a max number of steps. When the agent reaches the max step and has not found the target state, this two-pin net is regarded as a failure routing in this episode.

4.4.5. Loss Function. We can use the Mean Square Error (MSE) to calculate the Euclidean distance between the predicted Q-value and the target Q-value. The loss function is shown in Formula (11). The loss function obtains the gradient equation by deriving network parameters ω and uses the Adam optimizer to perform stochastic gradient descent [34]:

$$J(\omega) = \frac{1}{2m} \sum_{i=1}^{m} [y_i - Q(s_i, a_i ; \omega)]^2, \tag{11}$$

where m is the number of training samples and y_i is calculated by Formula (8).

4.5. Concurrent Training. Section 4.4 mentioned that by changing the reward function to guide the agent to share wires within the net, it could be found that the relationship among two-pin nets inside a net cooperates. However, the two-pin nets of different nets compete for edge capacity. In serial training (the nets are routed in a given order), the nets that are routed earlier are regarded as obstacles by the nets that are routed later. As a result, the later nets are more difficult to route than the previous nets.

In response to the above problems, we propose a concurrent training method. In a given environment, the nets are sorted in descending order of the number of pins, and then, the ordered nets are stored in a concurrent queue. Suppose that the number of nets is N, the value of max step is maxStep, the number of two-pin nets of Net i is n^i_{twopins}, and the max number of executable steps that Net i can perform is $n^i_{\text{twopins}} \times \text{maxStep}$. An episode is divided into multiple time slices in concurrent training. A time slice allocates 1

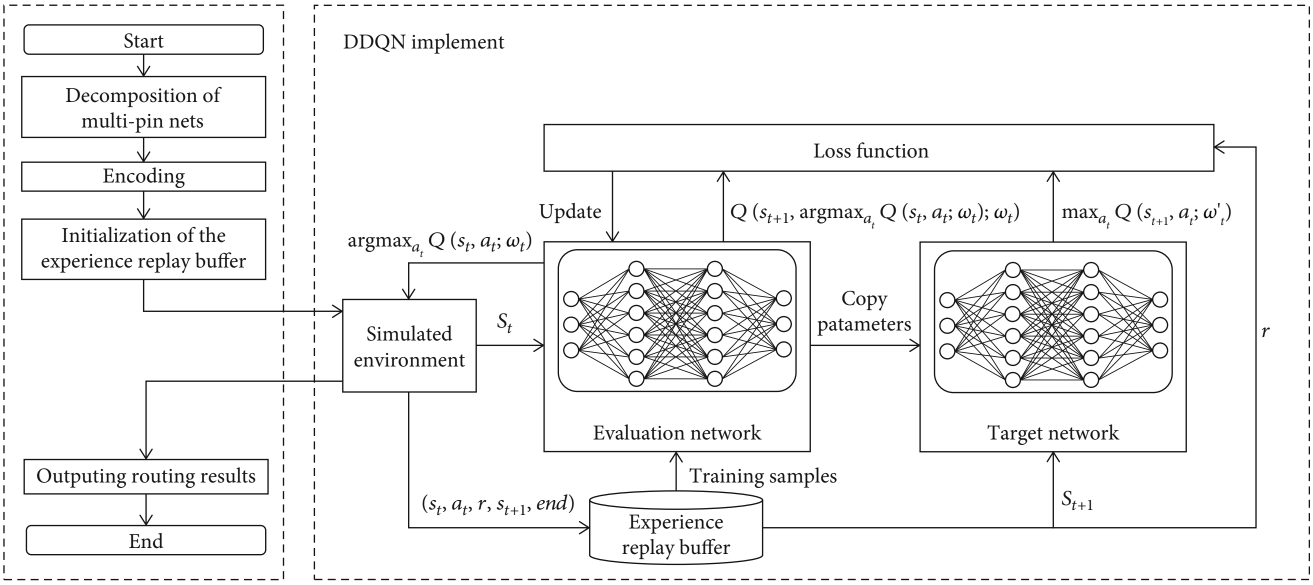

Start
Decomposition of multi-pin nets
Encoding
Initialization of the experience replay buffer
Outputing routing results
End
DDQN implement
Loss function
Update
$Q\,(s_{t+1}, \text{argmax}_{a_t} Q\,(s_t, a_t; \omega_t); \omega_t)$
$\max_{a_t} Q\,(s_{t+1}, a_t; \omega'_t)$
$\text{argmax}_{a_t} Q\,(s_t, a_t; \omega_t)$
Simulated environment
S_t
Copy patameters
Evaluation network
Target network
r
Training samples
$(s_t, a_t, r, s_{t+1}, end)$
Experience replay buffer
S_{t+1}

Figure 6: Algorithm flow chart.

step for each net in the concurrent queue (if the max step of Net i has been exhausted or Net i has been successfully routed, Net i will no longer enter the concurrent queue in the current episode). Through concurrent training, the priorities of all nets in the queue are increased to the same level. The performance comparison of serial training and concurrent training will be discussed in detail in Section 5.

4.6. Algorithm Flow Chart and Complexity Analysis. Figure 6 is a flow chart, and Algorithm 1 is the pseudocode of the DDQN-based global router. Lines 1-4 are multipin net decomposition, encoding, network weight initialization, and experience replay buffer initialization, respectively. Lines 6-20 are network training. Line 7 is two-pin net traversing. Line 8 is the first state acquisition of a two-pin net. Lines 11-13 are path finding. Line 14 is experience storing. Lines 15-17 are experience sampling, loss acquisition, and gradient descent, respectively.

We explain the algorithm complexity of each step.

4.6.1. Decomposition of Multipin Nets. The first step of the algorithm is to decompose the multipin nets. We use the Prim algorithm to construct the MST. In a net, the algorithm randomly selects a pin as the root point of the spanning tree. Mark the root as visited and initialize the distance from the root to all other unvisited points. Then repeat the following steps: select a point from the unvisited points with the minimum distance to the spanning tree, and add the point to the spanning tree. This process uses a priority queue to store unvisited points, which is used to find the nearest unvisited point. Therefore, the time complexity of the Prim algorithm with the priority queue is $O(n \log n)$, where n is the number of pins in the net. In the global routing with m nets, the total time complexity is $O(\sum_{i=1}^{m} n_i \log n_i)$, where n_i is the number of pins of Net i.

```
1:  Decompose multi-pin nets with Prim algorithm
2:  Encode two-pin nets Prim algorithm
3:  Initialize Q-network with random weights
4:  Initialize experience replay buffer with $A^*$ router
5:  Network training:
6:  **for** episode : episodes **do**
7:    **for** two-pin net : two-pin nets **do** //Concurrent 到底是哪部分并行
8:      Get initial state $s_0$ code for two-pin net
9:      **for** $t = 1$ : max step **do**
10:       Eliminate redundant actions 具体是怎么eliminate的？
11:       With $\varepsilon -$ greedy policy, select an action $a_t$
12:       Take action $a_t$ in environment and get reward
          $r_t$ and state $s_{t+1}$
13:       Update routing information
14:       Store experience $< s_t, a_t, r_t, s_{t+1}, \text{end} > 15$
15:       Randomly sample training samples
16:       Set $y_j = r_j + \gamma Q^*(s_j, a(s_j\,;\omega)\,;\omega')$
17:       Perform a gradient descent step on MSE($y_j$,
          $Q(s_j, a_j\,;\omega)$)
18:     **end for**
19:   **end for**
20: **end for**
```

Algorithm 1: DDQN-based global router.

4.6.2. Encoding. The starting point of each two-pin net is coded. It only takes $O(1)$ time to convert a two-pin net into a 15-bit code. If the number of nets is m, then the time complexity is $O(m)$.

4.6.3. Initialization of the Experience Replay Buffer. This algorithm searches the paths of all two-pin nets through A^* algorithm for initialization. The path search space of a two-pin net is limited to the Minimum Bounding Box (MBB) to reduce time consumption. Next, we analyze the time complexity of this step. If an MBB of a two-pin net

Table 1: Wire length comparison between DQN-based and DDQN-based global routers.

No.	Grid size	Net number	Max step	Capacity	Wire length		Optimization rate
					DQN	DDQN	
1	8*8*2	20	2	3	167	165	1.20%
2	8*8*2	20	2	3	170	168	1.18%
3	8*8*2	20	5	4	264	257	2.65%
4	8*8*2	20	5	4	250	249	0.40%
5	8*8*2	40	2	4	Fail	277	—
6	8*8*2	40	2	4	Fail	293	—
7	16*16*2	40	2	3	480	467	2.71%
8	16*16*2	40	2	3	Fail	521	—
9	16*16*2	40	5	5	949	937	1.26%
10	16*16*2	40	5	5	806	801	0.62%
Average							1.43%

has k G-cells, then in the worst case, all k G-cells will be visited, where $k = a \times b \times h$ and a, b, and h are the length, width, and height of the MBB, respectively. A^* expands the G-cell with the maximum evaluation value $f(n)$ each time. This process requires a priority queue to store the G-cells with their evaluation value. The time consumption for the insertion, deletion, and update of the priority queue is $O(\log k)$. Therefore, in the worst case, the time complexity of A^* is $O(k \log k)$. In the case of t two-pin nets, the required time for this step is $O(\sum_{i=1}^{t} k_i \log k_i)$, where k_i is the MBB size of Net i.

最大

4.6.4. *DDQN Implement.* When the episode and the max step are huge, the time consumed by the gradient descent and parameter copy process can be regarded as a constant. In the worst case, each episode executes n_t steps, where n_t is the number of max steps, so the worst time complexity of DDQN is $O(1)$.

4.6.5. *Complexity of DDQN-Based Global Router.* The time complexity of the DDQN-based global router is determined by the most time-consuming step. With the constraints of the global routing model, there is $k_i \geq n_i$, so the most time-consuming step is the experience replay buffer initialization, and the total time complexity is $O(\sum_{i=1}^{t} k_i \log k_i)$.

5. Experimental Results

5.1. *Development Environment.* We use Python and TensorFlow machine learning libraries to implement this algorithm. The experimental environment is Intel Core i5-9500F CPU, 8.0 G memory, and Windows 10 operating system. The benchmarks to test this algorithm are automatically generated by the Benchmark Generator provided by [16]. The setting of hyperparameters is consistent with [16].

5.2. *Wire Length and Convergence Comparisons between DQN-Based and DDQN-Based Global Router.* In order to verify the enhanced performance of the DDQN-based global router after overcoming overestimation, we compare the two routers in terms of wire length and convergence.

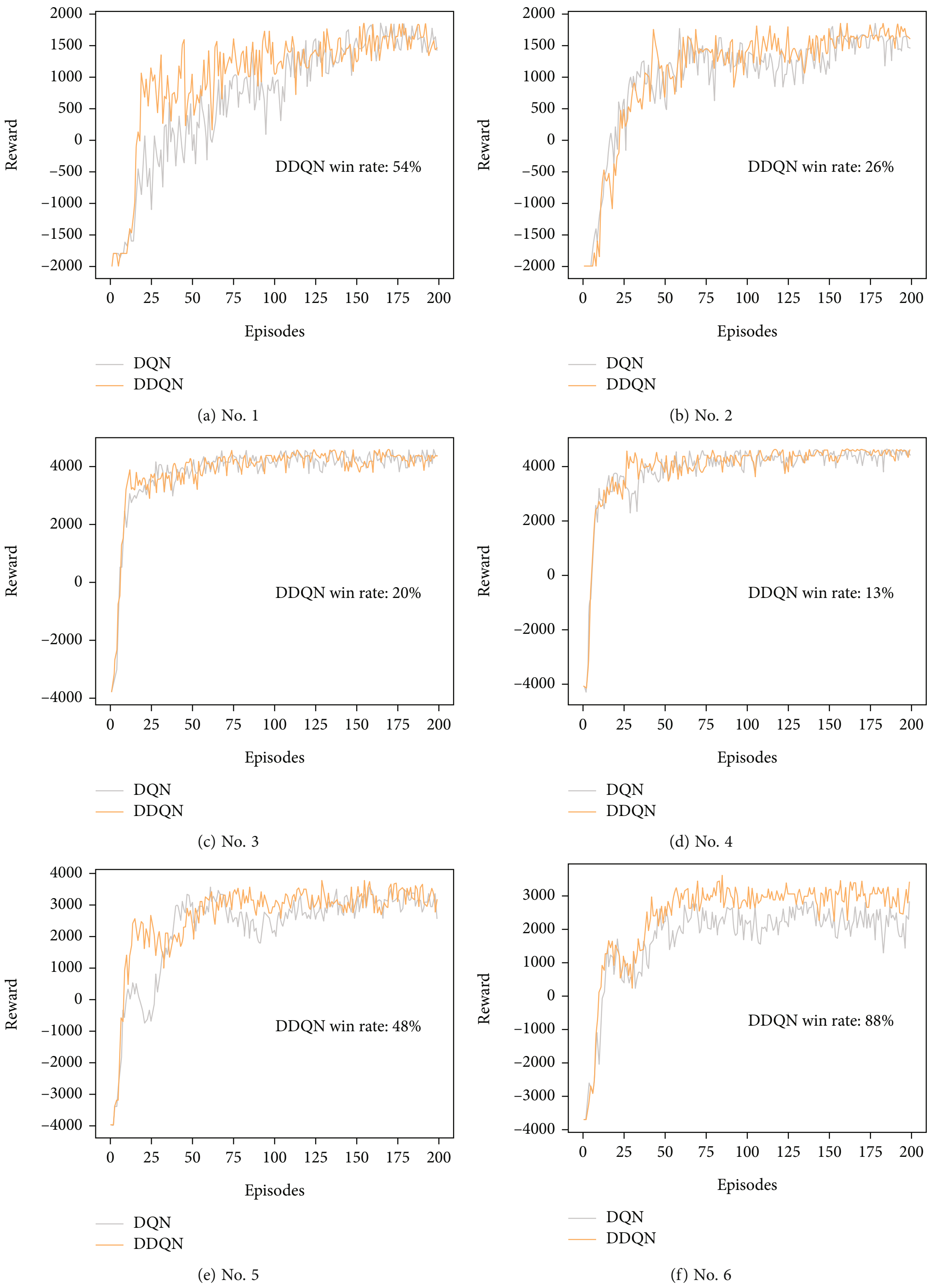
提升很小

5.2.1. *The Comparative Experiment of Wire Length.* As shown in Table 1, we generated ten benchmarks for the experiment. Benchmarks of the same scale have different pin distributions. For example, the benchmarks of No. 1 and No. 2 have the same grid size, number of nets and pins, and capacities of each edge, but they have different pin distributions. DQN and DDQN models are training with 200 episodes, and other parameters are the same. It can be seen that the DDQN router has achieved wire length reduction in all benchmarks. In the test results of No. 5, No. 6, and No. 8, the DQN router has slow convergence due to overestimation problems, and the DQN routing still failed (router failed to find path) after training. DDQN successfully passed the test in ten benchmarks.

感觉他的评价标准也不对，不是应该看收敛结果吗，收敛后都差不多

5.2.2. *The Comparative Experiment of Convergence.* Figure 7 presents the reward line charts that reflect the convergence speed of two routers. The grey line in the charts represents the DQN-based global router reward line, while the orange line represents the DDQN-based router. As shown in Figures 7(a)–7(j), each chart corresponds to the reward line chart of benchmarks 1 to 10, respectively, as listed in Table 1. By comparing the reward in each episode, the gap between the convergence performance of the two routers can be determined. It is worth noting that a higher reward value indicates a better routing result learned by the agent.

To quantify the superiority of the DDQN-based router over the DQN-based, the win rate is calculated using the formula $[(a - b)/a] \times 100\%$, where a is the number of times that the DDQN-based global router's reward value is greater than the DQN-based in all episodes and b is the number of times the DDQN-based global router's reward value is less than the DQN-based in all episodes. In the ten benchmarks, the DDQN-based global router exhibits better convergence performance than the DQN-based. Specifically, in the benchmarks corresponding to Figures 7(a) and 7(e)–7(h), the



(a) No. 1

(b) No. 2

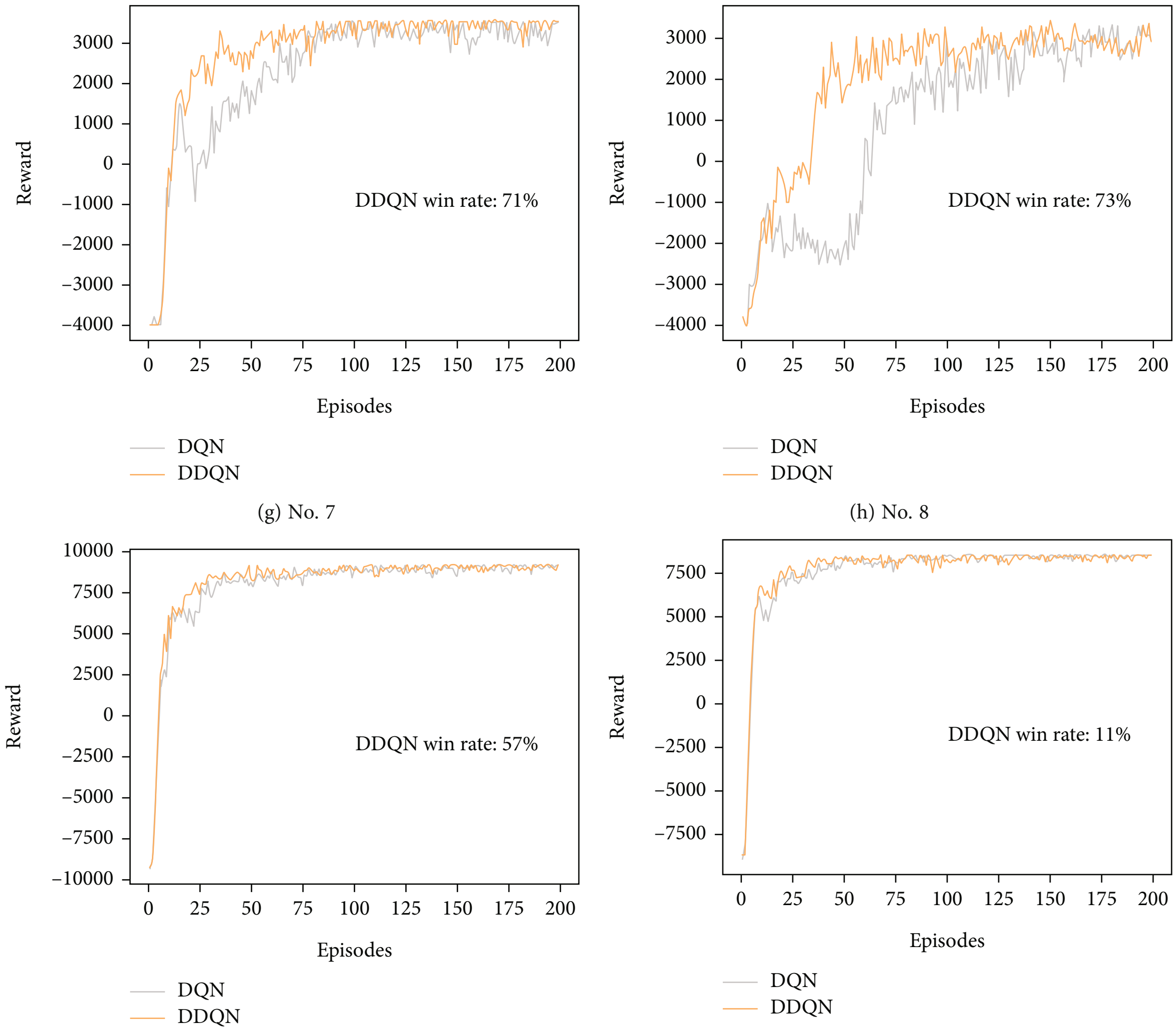
(c) No. 3

(d) No. 4

(e) No. 5

(f) No. 6

Figure 7: Continued.



(g) No. 7

(h) No. 8

(i) No. 9

(j) No. 10

FIGURE 7: Reward line chart of DQN-based and DDQN-based global routers.

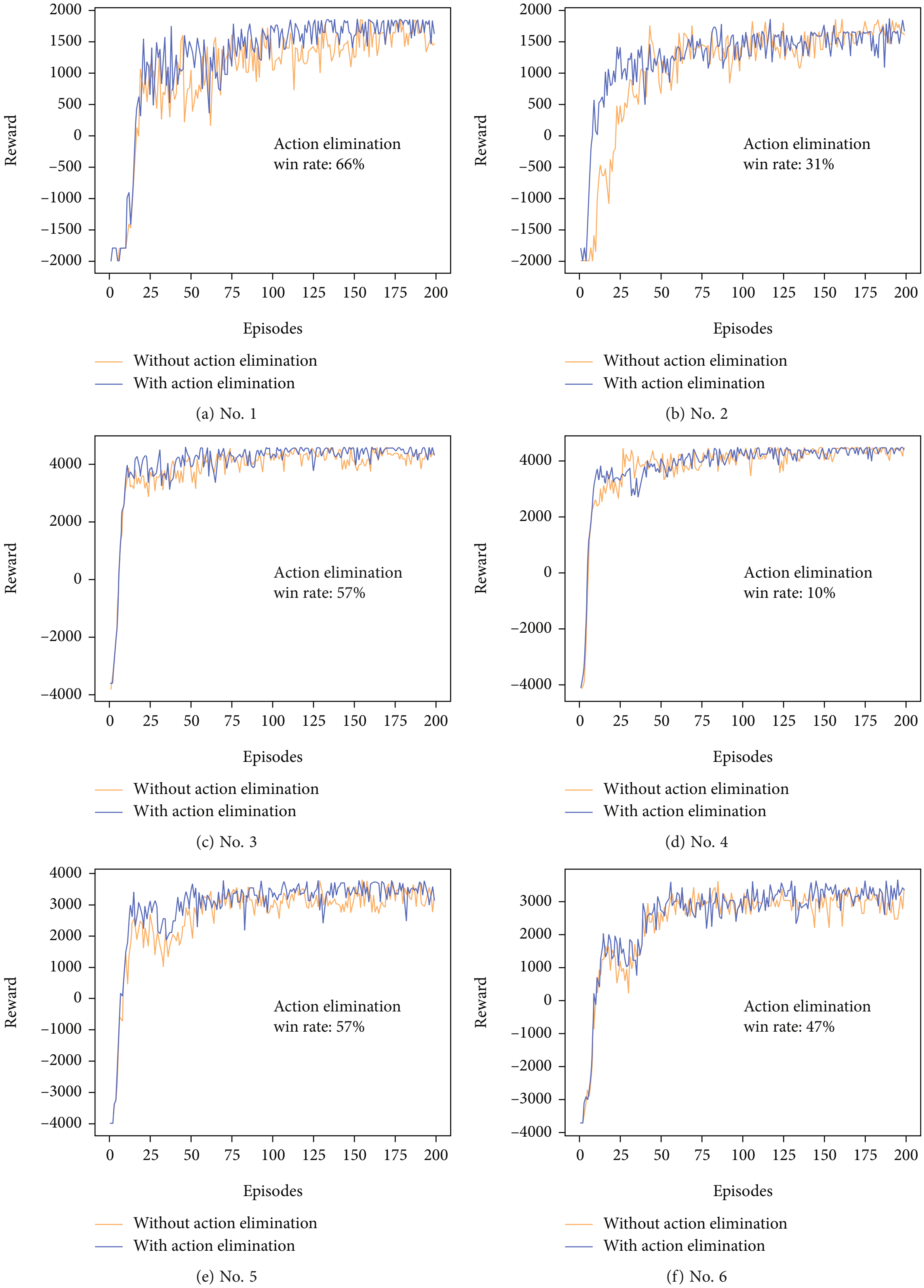
convergence performance of the DDQN-based global router is significantly higher than that of the DQN-based.

5.3. Verification of Action Elimination Method. The action elimination method prevents the agent from performing unnecessary actions in the environment and reduces the agent from learning redundant information. We compare the convergence speed of the DRL router before and after the action elimination method to verify the enhancement of the DRL router performance by this method.

We compare the performance of the DDQN-based global router with and without the action elimination method. Figures 8(a)–8(j) correspond to the line charts of the reward value of benchmarks 1 to 10 in Table 1. The orange line in Figure 8 is the reward value of the DDQN-based global router, and the blue line is the reward value of the DDQN-based global router with the action elimination method. From the experimental results, by eliminating unnecessary behaviours, the convergence speed of the router is optimized in ten benchmarks. Therefore, it is proved that this method can effectively enhance the convergence speed of the DDQN router.

5.4. Routability Comparison between Serial Training and Concurrent Training. The concurrent training method eliminates the unfairness caused by serial training, and the later nets can compete with the earlier nets for routing resources. The failure rates of later nets are reduced by overcoming the unfairness, thereby increasing the routability. We set up two experimental comparisons to verify that concurrent training can increase the routability of the DDQN-based global router: (1) compare the value of episode of the first successful routing in 200 episodes, as shown in Figure 9(a), and (2) compare the number of successful routing times in 200 episodes, as shown in Figure 9(b).

Figure 9(a) is a bar chart of the episode value for the first successful serial and concurrent training routing. The green bar represents serial training, and the purple represents concurrent training. It can be seen that in the ten benchmarks, concurrent training can learn a successful routing solution



(a) No. 1

(b) No. 2

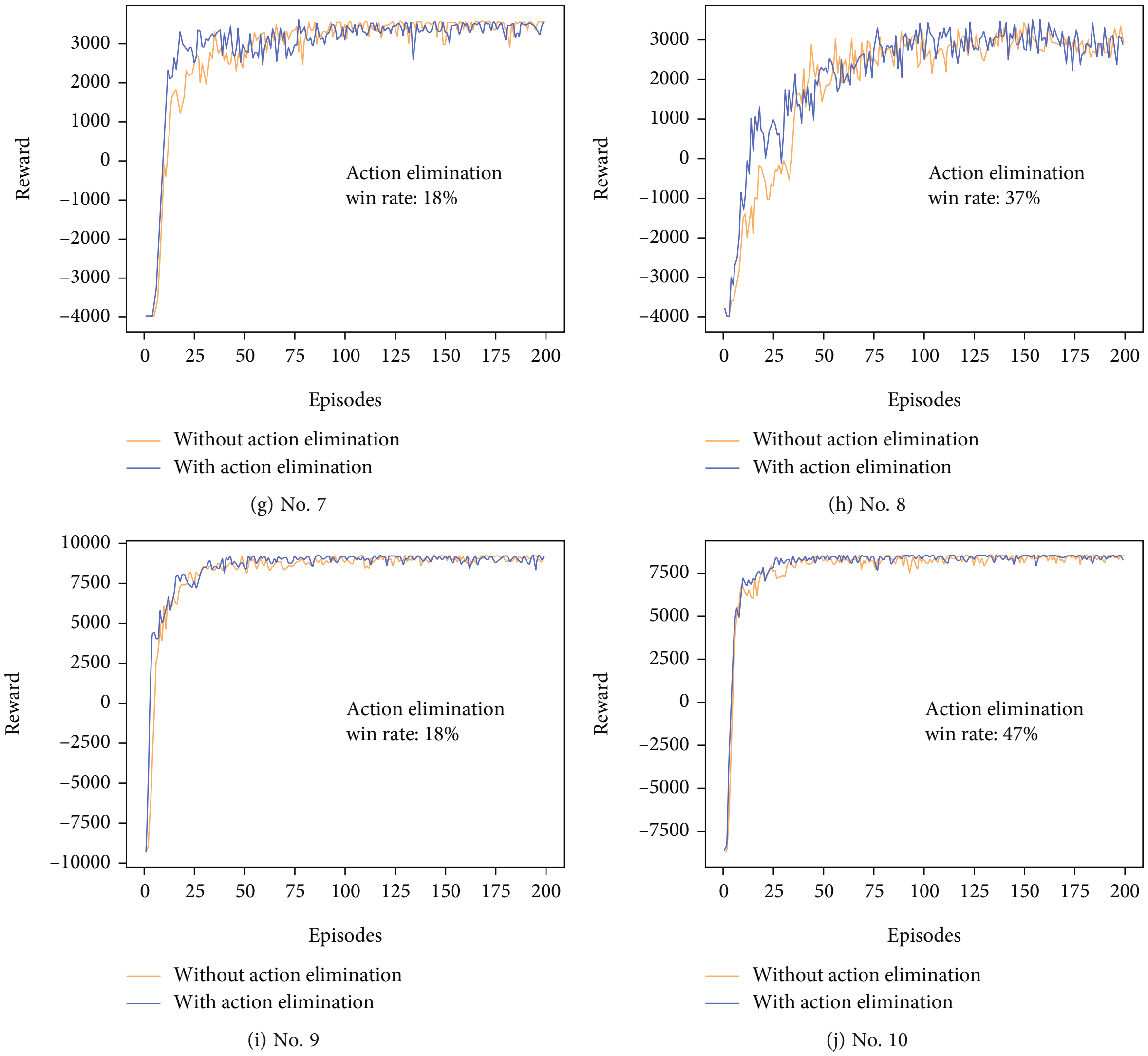
(c) No. 3

(d) No. 4

(e) No. 5

(f) No. 6

Figure 8: Continued.



(g) No. 7

(h) No. 8

(i) No. 9

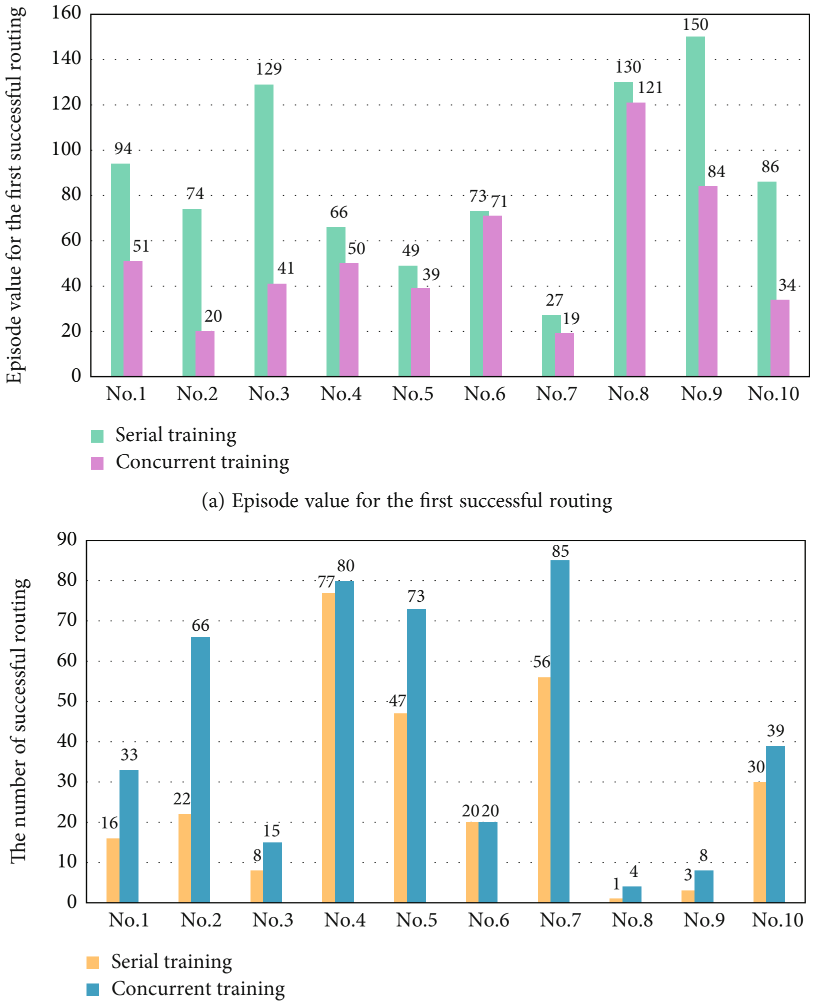
(j) No. 10

Figure 8: The enhanced performance brought by action elimination in ten benchmarks.

faster. Figure 9(b) is a bar chart of the number of successful routing in 200 episodes. The orange bar represents serial training, and the blue bar represents concurrent training. In the ten benchmarks, the number of successful routing of concurrent training is not less than that of serial training. These two experimental results prove that concurrent training can more reasonably allocate routing resources and significantly increase routability.

5.5. Routing Result Comparison of the Two Reward Functions. This part compares the performance of reward function 1 represented by Formula (9) and reward function 2 represented by Formula (10) in the DDQN-based global router. Reward function 2 focuses on sharing wires within the nets and the dispersion of the routing resources. Therefore, we compare the wire length and the congestion standard deviation of the routing solutions with the two reward functions.

The experimental results are shown in Table 2. The ten benchmarks are the same as in Table 1. The routing solutions with reward function 2 are better than reward function 1 in terms of wire length and congestion standard deviation. From the average of the results, it is possible to obtain 1.07% wire length and 5.71% congestion standard deviation optimization. Therefore, the experimental results can prove that reward function 2 is better than reward function 1. To explain the distribution of routing resources more intuitively, we have drawn the heat maps of the routing solutions of benchmark 9 in Figure 10. Figures 10(a) and 10(b) represent the first and second layers of reward function 2, respectively. Figures 10(c) and 10(d) represent the first and second layers of reward function 1, respectively. A square in the heat maps represents the remaining capacity of one edge. The lighter the square, the more demands of the edge. Compared with reward function 1, reward function 2 results in fewer edges with capacity $e_c < 2$. The experimental results have confirmed that reward function 2 both spreads the routing resources and reduces the wire length, which is in line with the original design intention. The DDQN-based global router in this paper will not overflow, so there will be no negative values in the heat maps.



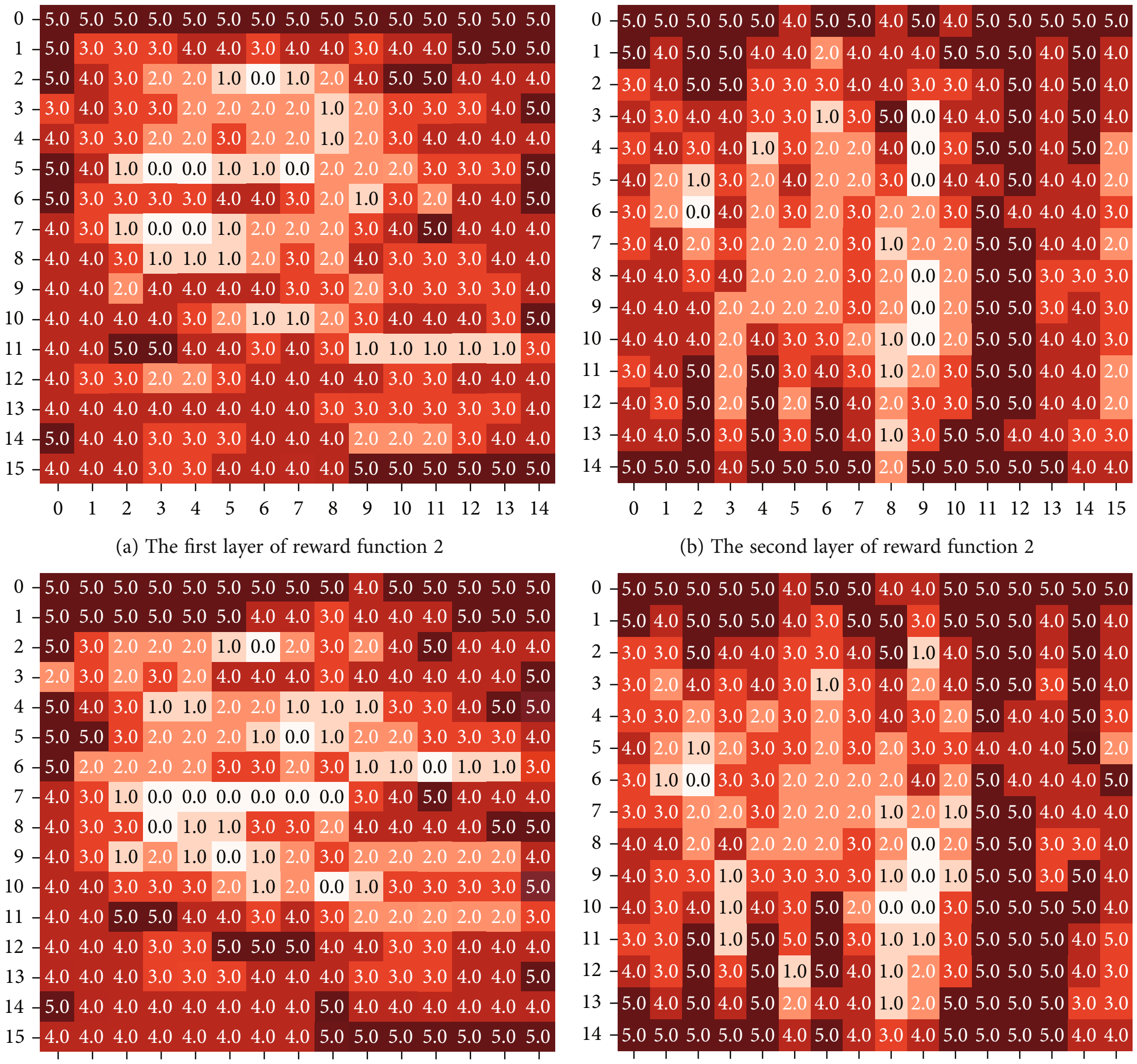
(a) Episode value for the first successful routing

(b) The number of successful routing

Figure 9: The episode value of the first successful routing and the number of successful routing of serial training and concurrent training.

Table 2: Comparison of wire length and congestion standard deviation of two reward functions.

No.	Wire length (WL)		Congestion standard deviation (Std)		Optimization rate (%)	
	Function 1	Function 2	Function 1	Function 2	WL	Std
1	165	165	0.95	0.85	0.00	10.53
2	168	165	0.94	0.94	1.79	0.00
3	257	248	1.30	1.12	3.50	7.44
4	249	245	1.23	1.16	0.40	5.69
5	277	274	1.31	1.23	1.08	2.38
6	293	290	1.32	1.23	1.02	6.82
7	467	466	0.83	0.81	0.21	4.71
8	521	504	0.95	0.89	3.26	6.32
9	937	934	1.36	1.29	0.32	7.19
10	801	794	1.16	1.09	0.87	6.03
Average					1.25	5.71



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0
1	5.0	3.0	3.0	3.0	4.0	4.0	3.0	4.0	4.0	3.0	4.0	4.0	5.0	5.0	5.0
2	5.0	4.0	3.0	2.0	2.0	1.0	0.0	1.0	2.0	4.0	5.0	5.0	4.0	4.0	4.0
3	3.0	4.0	3.0	3.0	2.0	2.0	2.0	2.0	1.0	2.0	3.0	3.0	3.0	4.0	5.0
4	4.0	3.0	3.0	2.0	2.0	3.0	2.0	2.0	1.0	2.0	3.0	4.0	4.0	4.0	4.0
5	5.0	4.0	1.0	0.0	0.0	1.0	1.0	0.0	2.0	2.0	2.0	3.0	3.0	3.0	5.0
6	5.0	3.0	3.0	3.0	3.0	4.0	4.0	3.0	2.0	1.0	3.0	2.0	4.0	4.0	5.0
7	4.0	3.0	1.0	0.0	0.0	1.0	2.0	2.0	2.0	3.0	4.0	5.0	4.0	4.0	4.0
8	4.0	4.0	3.0	1.0	1.0	1.0	2.0	3.0	2.0	4.0	3.0	3.0	3.0	4.0	4.0
9	4.0	4.0	2.0	4.0	4.0	4.0	4.0	3.0	3.0	2.0	3.0	3.0	3.0	3.0	4.0
10	4.0	4.0	4.0	4.0	3.0	2.0	1.0	1.0	2.0	3.0	4.0	4.0	4.0	3.0	5.0
11	4.0	4.0	5.0	5.0	4.0	4.0	3.0	4.0	3.0	1.0	1.0	1.0	1.0	1.0	3.0
12	4.0	3.0	3.0	2.0	2.0	3.0	4.0	4.0	4.0	4.0	3.0	3.0	4.0	4.0	4.0
13	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	3.0	3.0	3.0	3.0	3.0	3.0	4.0
14	5.0	4.0	4.0	3.0	3.0	3.0	4.0	4.0	4.0	2.0	2.0	2.0	3.0	4.0	4.0
15	4.0	4.0	4.0	3.0	3.0	4.0	4.0	4.0	4.0	5.0	5.0	5.0	5.0	5.0	5.0

(a) The first layer of reward function 2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	5.0	5.0	5.0	5.0	5.0	4.0	5.0	5.0	4.0	5.0	4.0	5.0	5.0	5.0	5.0	5.0
1	5.0	4.0	5.0	5.0	4.0	4.0	2.0	4.0	4.0	4.0	5.0	5.0	5.0	4.0	5.0	4.0
2	3.0	4.0	5.0	5.0	3.0	3.0	3.0	4.0	4.0	3.0	3.0	4.0	5.0	4.0	5.0	4.0
3	4.0	3.0	4.0	4.0	3.0	3.0	1.0	3.0	5.0	0.0	4.0	4.0	5.0	4.0	5.0	4.0
4	3.0	4.0	3.0	4.0	1.0	3.0	2.0	2.0	4.0	0.0	3.0	5.0	5.0	4.0	5.0	2.0
5	4.0	2.0	1.0	3.0	2.0	4.0	2.0	2.0	3.0	0.0	4.0	4.0	5.0	4.0	4.0	2.0
6	3.0	2.0	0.0	4.0	2.0	3.0	2.0	3.0	2.0	2.0	3.0	5.0	4.0	4.0	4.0	3.0
7	3.0	4.0	2.0	3.0	2.0	2.0	2.0	3.0	1.0	2.0	2.0	5.0	5.0	4.0	4.0	2.0
8	4.0	4.0	3.0	4.0	2.0	2.0	2.0	3.0	2.0	0.0	2.0	5.0	5.0	3.0	3.0	3.0
9	4.0	4.0	4.0	2.0	2.0	2.0	2.0	3.0	2.0	0.0	2.0	5.0	5.0	3.0	4.0	3.0
10	4.0	4.0	4.0	2.0	4.0	3.0	3.0	2.0	1.0	0.0	2.0	5.0	5.0	4.0	4.0	3.0
11	3.0	4.0	5.0	2.0	5.0	3.0	4.0	3.0	1.0	2.0	3.0	5.0	5.0	4.0	4.0	2.0
12	4.0	3.0	5.0	2.0	5.0	2.0	5.0	4.0	2.0	3.0	3.0	5.0	5.0	4.0	4.0	2.0
13	4.0	4.0	5.0	3.0	5.0	3.0	5.0	4.0	1.0	3.0	5.0	5.0	4.0	4.0	3.0	3.0
14	5.0	5.0	5.0	4.0	5.0	5.0	5.0	5.0	2.0	5.0	5.0	5.0	5.0	5.0	4.0	4.0

(b) The second layer of reward function 2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	4.0	5.0	5.0	5.0	5.0	5.0
1	5.0	5.0	5.0	5.0	5.0	5.0	4.0	4.0	3.0	4.0	4.0	4.0	5.0	5.0	5.0
2	5.0	3.0	2.0	2.0	2.0	1.0	0.0	2.0	3.0	2.0	4.0	5.0	4.0	4.0	4.0
3	2.0	3.0	2.0	3.0	2.0	4.0	4.0	4.0	3.0	4.0	4.0	4.0	4.0	4.0	5.0
4	5.0	4.0	3.0	1.0	1.0	2.0	2.0	1.0	1.0	1.0	3.0	3.0	4.0	5.0	5.0
5	5.0	5.0	3.0	2.0	2.0	2.0	1.0	0.0	1.0	2.0	2.0	3.0	3.0	3.0	4.0
6	5.0	2.0	2.0	2.0	2.0	3.0	3.0	2.0	3.0	1.0	1.0	0.0	1.0	1.0	3.0
7	4.0	3.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	4.0	5.0	4.0	4.0	4.0
8	4.0	3.0	3.0	0.0	1.0	1.0	3.0	3.0	2.0	4.0	4.0	4.0	4.0	5.0	5.0
9	4.0	3.0	1.0	2.0	1.0	0.0	1.0	2.0	3.0	2.0	2.0	2.0	2.0	2.0	4.0
10	4.0	4.0	3.0	3.0	3.0	2.0	1.0	2.0	0.0	1.0	3.0	3.0	3.0	3.0	5.0
11	4.0	4.0	5.0	5.0	4.0	4.0	3.0	4.0	3.0	2.0	2.0	2.0	2.0	2.0	3.0
12	4.0	4.0	4.0	3.0	3.0	5.0	5.0	5.0	4.0	4.0	3.0	3.0	4.0	4.0	4.0
13	4.0	4.0	4.0	3.0	3.0	3.0	4.0	4.0	4.0	3.0	3.0	3.0	4.0	4.0	5.0
14	5.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	5.0	4.0	4.0	4.0	4.0	4.0	4.0
15	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0

(c) The first layer of reward function 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	5.0	5.0	5.0	5.0	5.0	4.0	5.0	5.0	4.0	4.0	5.0	5.0	5.0	5.0	5.0	5.0
1	5.0	4.0	5.0	5.0	5.0	4.0	3.0	5.0	5.0	3.0	5.0	5.0	5.0	4.0	5.0	4.0
2	3.0	3.0	5.0	4.0	4.0	3.0	3.0	4.0	5.0	1.0	4.0	5.0	5.0	4.0	5.0	4.0
3	3.0	2.0	4.0	3.0	4.0	3.0	1.0	3.0	4.0	2.0	4.0	5.0	5.0	3.0	5.0	4.0
4	3.0	3.0	2.0	3.0	2.0	3.0	2.0	3.0	4.0	3.0	2.0	5.0	4.0	4.0	5.0	3.0
5	4.0	2.0	1.0	2.0	3.0	3.0	2.0	3.0	2.0	3.0	3.0	4.0	4.0	4.0	5.0	2.0
6	3.0	1.0	0.0	3.0	3.0	2.0	2.0	2.0	2.0	4.0	2.0	5.0	4.0	4.0	4.0	5.0
7	3.0	3.0	2.0	2.0	3.0	2.0	2.0	2.0	1.0	2.0	1.0	5.0	5.0	4.0	4.0	4.0
8	4.0	4.0	2.0	4.0	2.0	2.0	2.0	3.0	2.0	0.0	2.0	5.0	5.0	3.0	3.0	4.0
9	4.0	3.0	3.0	1.0	3.0	3.0	3.0	3.0	1.0	0.0	1.0	5.0	5.0	3.0	5.0	4.0
10	4.0	3.0	4.0	1.0	4.0	3.0	5.0	2.0	0.0	0.0	3.0	5.0	5.0	5.0	5.0	4.0
11	3.0	3.0	5.0	1.0	5.0	5.0	5.0	3.0	1.0	1.0	3.0	5.0	5.0	5.0	4.0	5.0
12	4.0	3.0	5.0	3.0	5.0	1.0	5.0	4.0	1.0	2.0	3.0	5.0	5.0	5.0	4.0	3.0
13	5.0	4.0	5.0	4.0	5.0	2.0	4.0	4.0	1.0	2.0	5.0	5.0	5.0	5.0	3.0	3.0
14	5.0	5.0	5.0	5.0	5.0	4.0	5.0	4.0	3.0	4.0	5.0	5.0	5.0	5.0	4.0	4.0

(d) The second layer of reward function 1

Figure 10: Heat maps of routing solutions of benchmark 9.

5.6. Test of Conjoint Optimization of DDQN-Based Global Router. The main advantage of the DRL-based global router is that it can consider subproblems jointly. We regard the routing of a two-pin net as a subproblem; the DRL-based global router can optimize the routing scheme of the current subproblem according to the global situation. The heuristic algorithm does not possess this feature. [16] verifies the conjoint optimization capability of the DRL-based global router by comparing the sequential A^*-based global router, so we use the same method for verification.

Given a series of two-pin nets, use sequential A^*-based and DDQN-based global routers for routing. A^* relies on the valuation function $f(n) = g(n) + h(n)$ to find a way and always chooses the minimum valuation for expansion. When routing one of the many two-pin nets, A^* can only know the routing results that have been completed, completely neglecting to reserve routing resources for incompletely routed two-pin nets. It is worth noting that no matter how large the cost is set for the cost function $g(n)$, the overflow of the routing solution cannot be avoided in sequence A^*. The DDQN-based global router learns historical experience through the experience replay mechanism, and these experiences are global. Therefore, every step of DDQN is determined through historical experience, and this mechanism manufactures its conjoint optimization capabilities.

The results of the sequence A^*-based and DDQN-based global routers are shown in Table 3. We should note that the primary optimization goal of global routing is the overflow. The routing result with overflow will make the chip impossible to manufacture. In the test of ten benchmarks, the routing results of sequence A^* have overflowed, and the number of overflows is large. In contrast, the DDQN-based global router has obtained no overflow routing solutions in ten benchmarks. The router needs to bypass the crowded areas to avoid overflow, which will produce a large amount of wire length. Under the conjoint optimization of DDQN, the

Table 3: Comparison of wire length and overflow of two routers.

No.	Wire length (WL)		Overflow (OF)		Optimization rate (%)	
	A^*-based	DDQN-based	A^*-based	DDQN-based	WL	OF
1	163	165	27	0	-1.23	100
2	159	165	19	0	-3.77	100
3	243	248	16	0	-2.06	100
4	249	245	12	0	1.60	100
5	272	274	22	0	-0.74	100
6	277	290	32	0	-4.69	100
7	464	466	8	0	-0.43	100
8	496	504	19	0	-1.61	100
9	932	934	16	0	-0.21	100
10	787	794	14	0	-0.89	100
Average					-1.40	100

routing among the subproblems is coordinated to reduce the wire length generated by bypassing. We can see from the results that although the wire length of the DDQN-based global router is decreased by 1.40% compared to sequence A^* on average, the overflow problem is fundamentally solved.

6. Conclusions

By combining four effective optimization strategies, this paper proposes a high-quality DRL-based global routing algorithm. First of all, to avoid the overestimation problem caused by Q-learning, a better-performance DDQN-based global router is proposed. Secondly, an action elimination method is proposed to reduce the redundant information learned by the agent and enhance the convergence speed of the DDQN-based global router. Thirdly, to solve the problem of unfair distribution of routing resources, a concurrent training method is proposed. Through the experimental results, the router with a concurrent training method can obtain successful routing solutions faster and increase the routability. In a given 200 episodes, more successful routing solutions are obtained than serial training. Fourthly, to reduce the wire length and decentralize the routing resources of the solutions, a new reward function is designed in this paper. In the experimental comparison, the wire length and the standard deviation of the congestion have been optimized.

In conclusion, four optimization methods proposed in this paper can further enhance the performance of the DRL-based global router, which can achieve superior performance compared to the heuristic method and DRL-based global router, so as to obtain high-quality global routing results without overflow. In future work, we will study the multiagent reinforcement learning method for the global routing problem.

Data Availability

The source data used in the study is available on the following website: https://github.com/YangLiliang/DRL-for-Global-Routing.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The authors would like to thank H. Liao et al. of the Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA. This work was partially supported by the National Natural Science Foundation of China under Grant No. 61877010.

References

[1] J. Cohoon, J. Kairo, and J. Lienig, “Evolutionary algorithms for the physical design of VLSI circuits,” in *Advances in Evolutionary Computing*, pp. 683–711, Springer, 2003.

[2] M. R. Kramer, “The complexity of wirerouting and finding minimum area layouts for arbitrary VLSI circuits,” in *Advances in Computing Research*, vol. 2, pp. 129–146, JAI Press Inc, 1984.

[3] H. Jiang and S. S. Sapatnekar, “A survey on multi-net global routing for integrated circuits,” *Integration*, vol. 31, no. 1, pp. 1–49, 2001.

[4] H. Liao, Q. Dong, X. Dong et al., “Attention routing: track-assignment detailed routing using attention-based reinforcement learning,” in *Proceedings of the ASME 2020 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. Volume 11A: 46th Design Automation Conference (DAC)*, Virtual, Online, 2020.

[5] K. Namba, N. Takashina, and H. Ito, “Design for delay measurement aimed at detecting small delay defects on global routing resources in FPGA,” *IEICE Transactions on Information and Systems*, vol. E96.D, no. 8, pp. 1613–1623, 2013.

[6] S. Held, D. Muller, D. Rotter, R. Scheifele, V. Traub, and J. Vygen, “Global routing with timing constraints,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 406–419, 2018.

[7] T. Koide, T. Suzuki, S. I. Wakabayashi, and N. Yoshida, “An efficient timing-driven global routing method for standard cell layout,” *IEICE Transactions on Information and Systems*, vol. 79, no. 10, pp. 1410–1418, 1996.

[8] M. A. Wiering and M. Van Otterlo, “Reinforcement learning,” in *Adaptation, Learning, and Optimization*, vol. 12, no. 3, 2012Springer, 2012.

[9] N. Xie, H. Hachiya, and M. Sugiyama, “Artist agent: a reinforcement learning approach to automatic stroke generation in oriental ink painting,” *IEICE Transactions on Information and Systems*, vol. E96.D, no. 5, pp. 1134–1144, 2013.

[10] V. Mnih, K. Kavukcuoglu, D. Silver et al., “Playing atari with deep reinforcement learning,” 2013, https://arxiv.org/abs/1312.5602.

[11] J. Clifton and E. Laber, “Q-learning: theory and applications,” *Annual Review of Statistics and Its Application*, vol. 7, no. 1, pp. 279–301, 2020.

[12] J. Barceló, H. Grzybowska, and S. Pardo, “Vehicle routing and scheduling models, simulation and city logistics,” in *Dynamic Fleet Management*, pp. 163–195, Springer, 2007.

[13] R. Bellman, “A Markovian decision process,” *Indiana University Mathematics Journal*, vol. 6, no. 4, pp. 679–684, 1957.

[14] W.-H. Liu, W.-C. Kao, Y.-L. Li, and K.-Y. Chao, “NCTUGR 2.0: multithreaded collision-aware global routing with bounded-length maze routing,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 5, pp. 709–722, 2013.

[15] H. van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-learning,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[16] H. Liao, W. Zhang, X. Dong, B. Poczos, K. Shimada, and L. B. Kara, “A deep reinforcement learning approach for global routing,” *Journal of Mechanical Design*, vol. 142, no. 6, article 061701, 2020.

[17] C. J. Alpert, D. P. Mehta, and S. S. Sapatnekar, *Handbook of Algorithms for Physical Design Automation*, CRC Press, 2008.

[18] Z. Cao, T. T. Jing, Y. H. Jinjun Xiong, Z. Feng, L. He, and X.-L. Hong, “Fashion: a fast and accurate solution to global routing problem,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 4, pp. 726–737, 2008.

[19] M. D. Moffitt, “MaizeRouter: engineering an effective global router,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 11, pp. 2017–2026, 2008.

[20] K.-R. Dai, W.-H. Liu, and Y.-L. Li, “NCTU-GR: efficient simulated evolution-based rerouting and congestion-relaxed layer assignment on 3-D global routing,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 3, pp. 459–472, 2012.

[21] G. Liu, X. Zhang, W. Guo et al., “Timing-aware layer assignment for advanced process technologies considering via pillars,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 6, pp. 1957–1970, 2022.

[22] C. Albrecht, “Global routing by new approximation algorithms for multicommodity flow,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 5, pp. 622–632, 2001.

[23] T.-H. Wu, A. Davoodi, and J. T. Linderoth, “GRIP: scalable 3D global routing using integer programming,” in *Proceedings of the 46th Annual Design Automation Conference*, pp. 320–325, San Francisco, California, 2009.

[24] C. Chu and Y.-C. Wong, “FLUTE: fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 70–83, 2008.

[25] S. Ghosh, S. Nath, R. Biswas, P. Venkateswaran, J. K. Sing, and S. K. Sarkar, “PSO variants and its comparison with Firefly algorithm in solving VLSI global routing problem,” in *2018 IEEE Electron Devices Kolkata Conference (EDKCON)*, pp. 513–518, Kolkata, India, 2018.

[26] G. Liu, X. Huang, W. Guo, Y. Niu, and G. Chen, “Multilayer obstacle-avoiding X-architecture Steiner minimal tree construction based on particle swarm optimization,” *IEEE Transactions on Cybernetics*, vol. 45, no. 5, pp. 1003–1016, 2015.

[27] X. Chen, G. Liu, N. Xiong, S. Yaru, and G. Chen, “A survey of swarm intelligence techniques in VLSI routing problems,” *IEEE Access*, vol. 8, pp. 26266–26292, 2020.

[28] G. Liu, W. Zhu, X. Saijuan, Z. Zhuang, Y.-C. Chen, and G. Chen, “Efficient VLSI routing algorithm employing novel discrete PSO and multi-stage transformation,” *Journal of Ambient Intelligence and Humanized Computing*, 2020.

[29] G. Liu, L. Yang, X. Saijuan, Z. Li, Y.-C. Chen, and C.-H. Chen, “X-architecture Steiner minimal tree algorithm based on multi-strategy optimization discrete differential evolution,” *PeerJ Computer Science*, vol. 7, article e473, 2021.

[30] G. Liu, X. Chen, R. Zhou, X. Saijuan, Y.-C. Chen, and G. Chen, “Social learning discrete particle swarm optimization based two-stage X-routing for IC design under Intelligent Edge Computing architecture,” *Applied Soft Computing*, vol. 104, article 107215, 2021.

[31] M. Gester, D. Müller, T. Nieberg, C. Panten, C. Schulte, and J. Vygen, “BonnRoute,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 18, no. 2, pp. 1–24, 2013.

[32] G. Liu, Z. Chen, Z. Zhuang, W. Guo, and G. Chen, “A unified algorithm based on HTS and self-adapting PSO for the construction of octagonal and rectilinear SMT,” *Soft Computing*, vol. 24, no. 6, pp. 3943–3961, 2020.

[33] G. Liu, Y. Zhu, X. Saijuan, Y.-C. Chen, and H. Tang, “PSO-based Power-Driven X-Routing algorithm in semiconductor design for predictive intelligence of IoT applications,” *Applied Soft Computing*, vol. 114, article 108114, 2022.

[34] Z. Zhang, “Improved Adam optimizer for deep neural networks,” in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pp. 1-2, Banff, AB, Canada, 2018.