Haiguang Liao
Department of Mechanical Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15213
e-mail: haiguanl@andrew.cmu.edu

Wentai Zhang
Department of Mechanical Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15213
e-mail: wentaiz@andrew.cmu.edu

Xuliang Dong
Department of Mechanical Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15213
e-mail: xuliangd@andrew.cmu.edu

Barnabas Poczos
Department of Machine Learning,
Carnegie Mellon University,
Pittsburgh, PA 15213
e-mail: bapoczos@cs.cmu.edu

Kenji Shimada
Department of Mechanical Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15213
e-mail: shimada@andrew.cmu.edu

Levent Burak Kara[1]
Department of Mechanical Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15213
e-mail: lkara@cmu.edu

A Deep Reinforcement Learning Approach for Global Routing

Global routing has been a historically challenging problem in the electronic circuit design, where the challenge is to connect a large and arbitrary number of circuit components with wires without violating the design rules for the printed circuit boards or integrated circuits. Similar routing problems also exist in the design of complex hydraulic systems, pipe systems, and logistic networks. Existing solutions typically consist of greedy algorithms and hard-coded heuristics. As such, existing approaches suffer from a lack of model flexibility and usually fail to solve sub-problems conjointly. As an alternative approach, this work presents a deep reinforcement learning method for solving the global routing problem in a simulated environment. At the heart of the proposed method is deep reinforcement learning that enables an agent to produce a policy for routing based on the variety of problems, and it is presented with leveraging the conjoint optimization mechanism of deep reinforcement learning. Conjoint optimization mechanism is explained and demonstrated in detail; the best network structure and the parameters of the learned model are explored. Based on the fine-tuned model, routing solutions and rewards are presented and analyzed. The results indicate that the approach can outperform the benchmark method of a sequential A method, suggesting a promising potential for deep reinforcement learning for global routing and other routing or path planning problems in general. Another major contribution of this work is the development of a global routing problem sets generator with the ability to generate parameterized global routing problem sets with different size and constraints, enabling evaluation of different routing algorithms and the generation of training datasets for future data-driven routing approaches.* [DOI: 10.1115/1.4045044]

Keywords: agent-based design, artificial intelligence, design automation, machine learning

1 Introduction

Keeping pace with Moore's law, integrated circuits (ICs) are becoming increasingly more sophisticated with the number of transistors in a unit area increasing exponentially over time [1,2]. More capable automatic design systems are needed to help engineers tasked with solving increasingly more challenging IC design problems. In the design flow of IC, global routing is a particularly critical and challenging stage in which the resources for routing (hence design constraints) in the chip design are allocated based on the preceding step of component placement and configurations of the chip to be designed [3]. The routing problem is then addressed in two stages, the first being the global routing step (the focus of this work), followed by detailed routing where the actual path of the wires and vias connecting those electronic components are decided.

Global routing has been a historically challenging problem in IC physical design for decades. Global routing involves a large and arbitrary number of *nets* to be routed, where each net may consist of many *pins* to be interconnected with wires. Even the simplest version of the problem where a single net with only two pins needs to be routed under the design constraints is an NP (nondeterministic polynomial)-complete problem [4]. In real settings, millions of components and nets may need to be integrated on a single chip, making the problem extremely challenging. Similar routing problems are encountered in other domains including routing-design of hydraulic systems [5], complex pipe system routing in ships [6], urban water distribution systems [7,8], and routing of city logistic services [9,10].

Owing to the difficulties of the problem and demand for tools for designing increasingly complicated IC, global routing has been one of the most active research areas in IC design [3,11–13]. However, current solutions rely primarily on heuristically driven greedy methods such as net ordering followed by sequential net routing [3], routing different areas of chip sequentially [14], and net ordering followed by force-directed routing [15]. These heuristics-based methods are primarily applicable with strong constraints on the problems to be solved. Besides, the unique hierarchical nature of global routing with net ordering and net decomposition makes solutions to standard graph or combinatorial problems such as the traveling salesman problem not readily applicable to global routing. As such, there remains potential to identify better solutions that can further minimize the objective functions of interest such as the total wirelength (WL) and edge overflow (OF) commonly utilized in IC design.

This work presents a deep reinforcement learning (DRL)-driven approach to IC global routing and analyzes its conjoint optimization mechanism to effectively address the unique challenges in IC global routing. DRL combines reinforcement learning and deep learning and has been successfully utilized in a variety of sequential decision making problems [16–21]. In our setting, both the state space (routing states) and the action space (routing decisions) are discrete and finite. Moreover, since a future routing state only depends on the current routing state and all state features are observable, we model the problem as a *Markov decision process* [22]. We

[1]Corresponding author.
Contributed by the Design Automation Committee of ASME for publication in the Journal of Mechanical Design. Manuscript received June 20, 2019; final manuscript received September 19, 2019; published online October 1, 2019. Assoc. Editor: James T. Allison.

choose a deep Q-network (DQN) as our fundamental RL algorithm to address the global routing problem.

At the heart of our approach is the observation that a properly trained *Q-network* within our DQN formulation can consider the nets and the pins to be routed conjointly with a single network and in a heuristics-free fashion, as opposed to attempting to route them sequentially with little or no backtracking. A single Q-network is trained and learned on the target problem of interest. When trained, it produces a "best" action for the dynamically evolving state of the routing by taking into account all the previous successful as well as unsuccessful net routing attempts that inform a reward function.

To the best of our knowledge, the presented work is the first attempt to formulate and solve IC global routing as a deep reinforcement learning problem. An IC global routing problem sets generator is also developed to automatically generate parameterized problems with different scales and constraints, enabling comparisons between algorithms and providing training, testing, and validation data for future data-driven approaches. We describe our DQN formulation together with the parametric studies of the best parameters of the algorithm and network structure. We compare our approach against solutions obtained by the commonly used A* search, which is sequentially applied to solve a set of nets. It is noted however that our approach, similar to previous approaches, does not guarantee global optimum.

2 Background and Related Work

2.1 Global Routing. Global routing allocates space resources on the chip to be designed by connecting the electronic components in a circuit coarsely based on a given placement solution [23]. It can be modeled as a grid graph $G(V, E)$, where each vertex v_i represents a rectangular region of the chip, so-called a global routing cell (Gcell) or a global routing tile, and each edge e_{ij} represents the boundary between v_i and v_j. In a given routing problem, IC design considerations impose constraints on the edges between neighboring global routing tiles. For this, each edge e_{ij} is assigned a capacity $c_{ij} \in \mathbf{Z}^+$ that indicates the maximum number of wire crossings that are permitted without penalty across that edge. In the final routing solution, if the number of wire crossings across e_{ij} exceed c_{ij}, this excess is represented as the overflow of the edge of_{ij} and the real-time capacity of e_{ij} becomes $-of_{ij}$. The routing problem takes as input an arbitrary number of nets, where each net has a number of pins with known layout coordinates to be interconnected.

Figure 1 shows an example of a real circuit and its corresponding grid graph. Global routing's goal is to find wiring paths that connect the pins inside the Gcells through $G(V, E)$ for all input nets. When connections are made through an edge whose capacity has been exhausted, the excess crossings are counted as the OF for that edge. The goal of global routing is to minimize: (1) the *congestion* in the final design, which is the total overflow accrued over all the edges and (2) the total WL, which is the sum of tile-to-tile connections emanating from all the nets used in the design. If two routing solutions achieve the same capacity constraint, the one with smaller wirelength will be a better solution. In most cases, a grid graph to be routed consists of more than one layer, and the connection from one cell to the neighboring cell in the layer above or below is called a via.

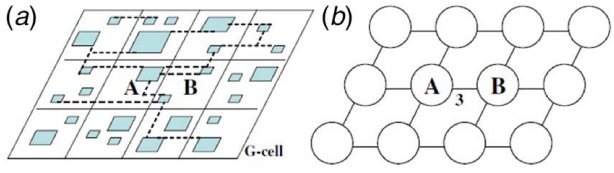

Fig. 1 Sample of a real circuit and corresponding grid graph for global routing [14]: (*a*) real circuit with G-cells and (*b*) grid graph for routing

In global routing, the nets are usually routed sequentially [3]. In routing each single net, there are often more than two pins to be connected, in which case methods such as the minimum spanning tree or rectilinear Steiner tree is used for decomposing a multiple-pin net problem into a set of two-pin connection problems [3,24–26]. Once a net is decomposed into a set of two-pin connection problems, search algorithms such as A* is used to find a solution to the two-pin problems. Once the set of two-pin problems of a net are solved, these solutions are merged into a solution for a single net.

Based on the above primary solution strategies, more advanced heuristic-based techniques including rip-up and reroute [27], force-directed routing, and region-wise routing [3] have been developed to increase the performance of global routing techniques. Additionally, a few machine learning-based techniques have been applied to global routing such as the prediction of routing congestion models [28] or routability prediction [29] with supervised models. However, these approaches only solve a part of the problem instead of providing a closed loop global routing solution. Also, to our knowledge, no prior work on global routing has investigated deep reinforcement learning as a possible new strategy.

2.2 Deep Reinforcement Learning. At the heart of reinforcement learning is the discovery of the optimal action-value function $Q^*(s, a)$ by maximizing the expected return starting from state s, taking action a, and then following policy π from there on that gives rise to the subsequent action-state configurations [30]

$$Q^*(s, a) = \max_{\pi} E[R_t | s_t = s, a_t = a, \pi] \tag{1}$$

The above optimal action-value function obeys the *Bellman equation* (Eq. (2)), wherein many reinforcement learning algorithms estimate the action-value function in an iterative manner. Obtaining updated value of action-value function following $Q_{i+1}(s, a) = E[r + \gamma \max_{a'} Q^*(s', a') | s, a]$, with expectation of current reward r and maximized future reward multiplied with a discount factor γ, given a current state s and an action a taken from that state.

$$Q^*(s, a) = E_{s' \sim \xi}[r + \gamma \max_{a'} Q^*(s', a') | s, a] \tag{2}$$

In actual implementation, a function approximator $Q(s, a; \theta)$ is used to estimate the optimal action-value function. In DRL, a deep neural network such as a convolutional neural network [16] or fully connected networks is used as the function approximator. The neural network used in the algorithm is often referred to as the Q-network. The Q-network is trained with a batch of sequences and actions by minimizing loss functions $L_i(\theta_i)$, which can be obtained by Eq. (3). Here, $y_i = E_{s' \sim \epsilon}[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a]$ is the target for iteration i and $\rho(s, a)$ is a probability distribution over the states and actions.

By differentiating the loss function with respect to its weights θ_i, the gradient equation is obtained. Based on the gradient, the network is trained with stochastic gradient descent. Once trained, actions can be obtained from the network following ϵ greedy strategy [16]:

$$L_i(\theta_i) = E_{s, a \sim \rho(.)}[(y_i - Q(s, a; \theta_i))^2] \tag{3}$$

$$\nabla_{\theta_i} L_i(\theta_i) = E[(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)) dQ] \tag{4}$$

$$dQ = \nabla_{\theta_i} Q(s, a; \theta_i) \tag{5}$$

3 Method

Figure 2 illustrates our DQN-based approach to global routing. First, the problem sets generator generates problems with a specified size, complexity, and constraints. All the problems generated are stored in separate text files. Following the convention in the global routing problems, the input text file contains all the information necessary to describe the problem, including the size of the grid, the edge capacity of the edges in the grid, the nets to be

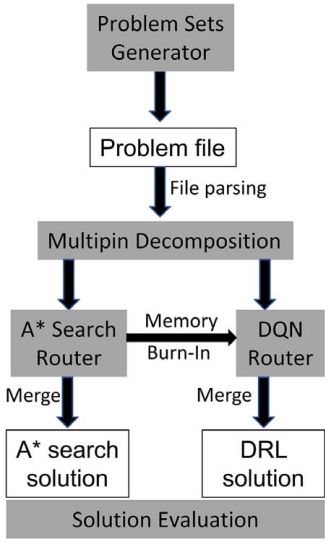

Fig. 2 Pipeline for solving global routing with DQN

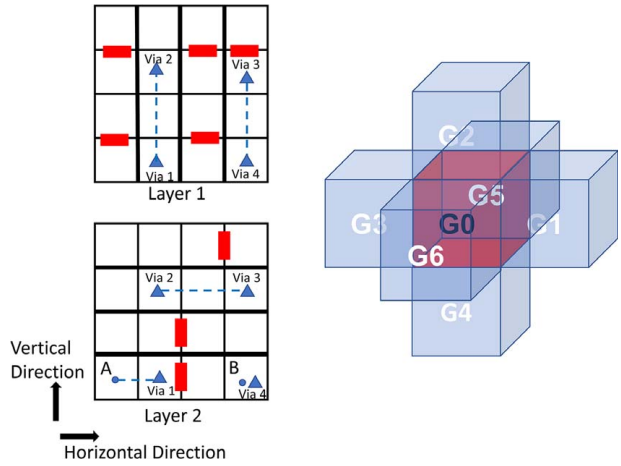

Fig. 3 Example configurations of the simulated global routing environment

routed, and for each net the spatial x, y, z coordinates of the pins that comprise the net. The input file is read in and parsed. Then, each net is further decomposed into a set of two-pin problems with the minimum spanning tree algorithm. A* router and DQN router are then used to solve the large set of two-pin problems. A* is executed first in order to provide burn-in memory for the DQN solver. After all two-pin problems emanating from all the nets are solved, the solutions belonging to the individual nets are merged. After a final solution is obtained for the entire set of nets, it is evaluated in light of the total congestion (sum of edge overflows, OF) and the total WL to assess the quality of that particular solution. All steps except solution evaluation are programed in PYTHON 3.6, and the evaluation is done using a separate module written in Perl. The details of the various steps are provided next.

3.1 Environmental Setup. Given a target chip routing problem, the text description input files consist of the chip information and all the nets to be routed. Chip information consists of grid size, edge capacities, and the reduced capacities of some of the edges owing to the pre-routed nets or other obstacles present on the chip. The nets information consists of a list of pin locations in each net. For a standard large-scale target problem, the number of pins belonging to a single net is not equal and may, in fact, vary from as few as two to more than a thousand. After the input file is read in and parsed, a simulated global routing environment that incorporates all the chip and nets information is created. The simulated environment is designed to have the following functions:

(1) For each net, decompose multi-pin problems into a set of two-pin problems. Feed routing solver with sets of two-pin routing problems, store and merge two-pin solutions to the final global routing solution.
(2) Provide reward feedback and observed sequence to DQN algorithms.
(3) Iteratively update edge capacities.

In general, the simulated global routing environment can be compared to a sequential version of a maze game, with edge capacities changing dynamically according to the thus-far routed path. Figure 3 shows an example of a simulated two-layer ($4\times4\times2$) problem environment with an illustrative routing solution for a two-pin problem. In Fig. 3, each layer consists of 16 Gcells (4×4), with layer 2 stacked right above layer 1. Bold edges have zero capacity, therefore the route can only be north–south (referred to as the vertical direction) in layer 1 and east–west (referred to as the horizontal direction) in layer 2 without generating OF. Red rectangles represent blocked edges owing to pre-routed wires or the existence of certain components. The two-pin problem is to generate a route from pin A to pin B, through different Gcells with the least WL and least OF possible. At each step, starting from a Gcell, there are six possible moves, as shown in Fig. 3 (right), which are going east (G1), west (G3), north (G5), south (G6), up (G2), and down (G4). In the simulated environment, actions are subject to boundary conditions and capacity constraints, therefore actual possible actions will vary and can be less than 6, depending on the environment settings of the problem. An illustrative solution to the problem is shown in Fig. 3. Once a routing step is taken, the capacity of the crossed edge changes accordingly.

In this work, $8\times8\times2$ problems consisting of different net numbers, pin numbers, and capacity are used to explore different settings and parameters. Results on larger problems are also shown to demonstrate scalability.

3.2 Global Routing Problem Sets Generator. In order to facilitate comparison of different algorithms of different scales and constraints, a global routing problem sets generator is developed with the ability to automatically generate a large problem database with user-specified *problem numbers, grid size, number of nets, number of pins each net, and capacity settings*. The generator imitates the features of industrial global routing problems by breaking down the capacity settings into two parts: normal capacity and reduced capacity. Normal capacity determines the capacity of edges on the horizontal direction (x direction) for the first layer and the vertical direction (y direction) on the second layer. Following the properties of the global routing problem, via capacity is set to be large enough in the simulated environment so that overflow will not occur in the via direction (z direction) and is therefore not specified in the problem file. Reduced capacity is described by setting a set of specific edges in the problem to prescribe capacity values. In our problem sets generator, reduced capacity can be set based on edge utilization conditions, which are statistical results showing the traffic conditions of all edges based on solutions given by A*. For instance, users can set a proportion of the most congested edges to generate a problem with very stringent constraints.

Figure 4 provides an example of edge utilization conditions of two generated problems with parameters specified (gridsize, $8\times8\times2$; number of nets, 50; max number of pins each net, 2; normal capacity, 3). The first problem (first row) does not have reduced capacity, while the second one (second row) got three of its most congested edge capacities reduced. The heat maps demonstrate the edge utilization condition of these two problems based on A* solutions, and the difference in the traffic condition of the problems can be observed. This is an example to demonstrate the problem sets generator's ability to generate problems with

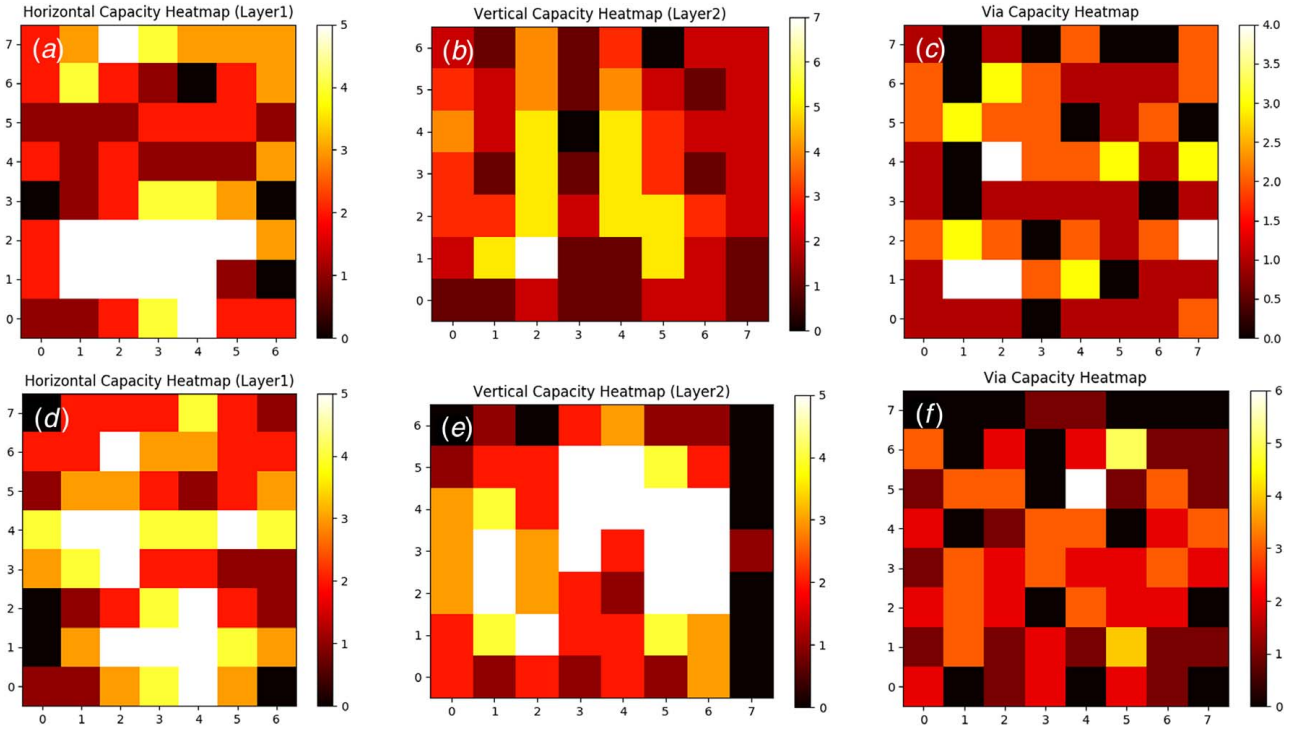

Fig. 4 Heat maps and edge utilization histogram of two global routing problems generated with global routing problem set generator. 8 × 8 × 2 benchmark with normal capacity setting: (*a*) via capacity heat map, (*b*) vertical capacity heat map, (*c*) horizontal capacity heat map, (*d*) edge utilization histogram; 8 × 8 × 2 benchmark with reduced capacity setting: (*e*) via capacity heat map, (*f*) vertical capacity heat map, (*g*) horizontal capacity heat map, and (*h*) edge utilization histogram. In these plots, darker cells correspond to less capacity utilization while lighter colored cells correspond to higher capacity utilization (i.e., more congestion) (number of nets, 50; max number of pins each net, 2; normal capacity, 3).

significant variability in not only general problem settings such as size and normal capacity but also detailed settings such as the amount and distribution of pin numbers and locations. More examples of heat maps based on different generated problem sets are provided in Appendix A. The source code for problem sets generator will be made publicly available.

3.3 Problem Breakdown and A* Search Solver. The solution obtained by A* serves as a benchmark against which the DRL solution can later be compared in terms of total WL and OF. In order to reduce the dimensions of the problem, a sequential A* approach is adopted, as shown in Fig. 2. All the nets in a single problem are dealt with sequentially, during which nets with multiple pins are decomposed with the minimum spanning tree method mentioned above.

Before DQN router starts to work, an A* search router is used to solve those two-pin problems. A* finds a path between the two pins (one of them taken as start S and the other taken as goal G), with the cost function defined as

$$f(n) = g(n) + h(n) \tag{6}$$

where n represents the time step, $g(n)$ represents the total path length from S to the current position, and $h(n)$ represents the future heuristic cost from the current position to G.

For each step, if there is no OF, the transition cost (incremental path length) is 1. In case of an OF, the cost becomes 1000. For $h(n)$, we use the Manhattan distance from the current position to G, which makes the heuristic admissible and thus the result optimal for that particular two-pin problem. However, it must be noted that the optimality of A* for the two-pin problems does not imply that the eventual final A* solution will be globally optimal, as the nets (and hence each net's resulting two-pin problems) are solved sequentially rather than conjointly. This issue is discussed in detail in Sec. 5.

As shown in Fig. 2, in addition to serving as a benchmark, A* search also provides burn-in memory for DRL to allow DRL to converge faster. However, using A* as burn-in memory is optional, as early random walks could also be used in place of A* for burn-in memory. Nonetheless, as will be shown later, our results indicate that using A* as burn-in for DRL allows DRL to converge much faster. An experimental approach is applied to determine the best for burn-in memory.

3.4 Deep Q-Networks Router Implementation. Using the same problem breakdown, a deep Q-network is utilized to solve all the two-pin problems in an iterative learning process. The framework of DQN is illustrated in Fig. 5. In this framework, the Q-network functions as an agent interacting with the environment. Specifically, for each two-pin problem, the environment provides the network with state information. Then, the agent evaluates the Q values of all the potential next states (q_1, q_2, …, q_6). Finally, based on an ϵ-greedy algorithm, an action is chosen and executed, altering the environment. The agent will have thus taken one "step" in the environment. A reward is calculated according to the new state and the edge capacity information is updated. In parallel, a replay buffer records each transition along the training process. These transitions are used for backpropagation when iteratively updating the weights in the Q-network. Further details of the DQN routing algorithm can be found in Appendix B. Critical elements of our approach are presented below.

State design: The state is defined as a 12-dimensional vector. The first three elements are the x, y, z coordinates of the current agent location in the environment. The fourth through the sixth elements encode the distance in the x, y, and z directions from the current agent location to the target pin location. The remaining six dimensions encode the capacity information of all the edges the agent is able to cross. This encoding scheme can be regarded as an admixture of the current state, the navigation, and the local capacity information.

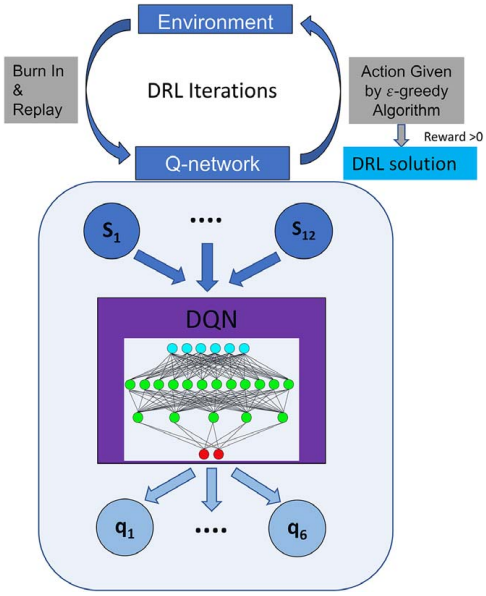

Fig. 5 Workflow of DQN-based router

Action space: The actions are represented with an integer from 0 to 5 corresponding to the direction of move from the current state.

Reward design: The reward is defined as a function of the chosen action and the next state $R(a, s')$. In our case

$$R(a, s') = \begin{cases} +100 & \text{if } s' \textit{ is the target pin} \\ -1 & \text{otherwise} \end{cases} \tag{7}$$

This design forces the agent to learn a path as short as possible since any unfruitful action will cause a decrement in the cumulative reward. Additionally, we limit the maximum number of steps the agent can take when solving each two-pin problem to be less than a maximum threshold $T^{\max}$, depending on the size of the problems to be solved. Following this reward design, the cumulative reward is always between $(100 - T^{\max})$ and 100 when the two-pin problem is solved, and $-T^{\max}$ otherwise. This scheme is a useful indicator to distinguish if the overall routing problem was successfully solved, or no feasible solution was found. Feasible $T^{\max}$ needs to be tuned according to the problem size.

Replay buffer and burn-in: The replay buffer is an archive of past transitions the agent experiences and is updated during training. A burn-in pre-process is introduced to fill in the replay buffer before training begins, which provides a basic knowledge of the environment to the network. In our case, the burn-in transitions are acquired during A* search. In each training iteration, a batch of transition records are randomly sampled from this replay buffer and used to update the network weights through back propagation.

As the Q-network is being trained, the two-pin problems that are solved need not be completed. That is, the network uses unsuccessful attempts (negative reward) at connecting the pin pairs in addition to the successful attempts (high, positive reward).

ϵ-greedy algorithm: In our case, the ϵ-greedy algorithm is utilized in the following form:

$$A(s) = \begin{cases} \text{a random action} & \text{with } \epsilon \text{ chance} \\ \text{argmax}_{a \in A}\ Q(s, a) & \text{with } 1 - \epsilon \text{ chance} \end{cases} \tag{8}$$

Network architecture: The Q-network consists of three fully connected layers with 32, 64, and 32 hidden units in each layer. Each layer is followed by a ReLU (Rectified Linear Unit) activation layer [31]. The input size is 12, which is the same as the number of elements in the designed state vector. The output size is 6, aligned with the number of possible next states.

Table 1 DQN model optimized parameters

Parameter	Value	Parameter	Value
Learning rate	1×10^{-4}	Batch size	32
Buffer size	50,000	Burn-in size	10,000
γ (discount factor)	1.0	Burn-in memory	A*-based
ϵ	0.05	Max episodes	5000

Training episodes: Given the nets and the associated extracted pin-pairs of each net, the Q-network gets updated iteratively over many cycles on the same target problem. A single passage of the entire set of pin-pairs, collected over all the nets, through the network constitutes an *episode*. The network is trained by processing the same pin-pairs over many episodes until convergence is reached. Once converged, the output of the Q-network allows the determination of the best action (as deemed by the network) for the given state of the problem.

Training parameters: These parameters are listed in Table 1, which only includes the optimized set of parameters. The model and the parameter tuning process are shown and discussed next.

Training specifications: The implementation of the proposed algorithm is completed with PYTHON and TENSORFLOW. The training process of an $8 \times 8 \times 2$ problem usually takes an hour on a workstation with an Intel Core i7-6850 CPU and an NVIDIA GeForce GTX 1080 Ti GPU.

4 Experiments

First, to make DQN models work, general parameter sets are selected for the batch size (32, 64, 128), learning rate (1×10^{-3}, 1×10^{-4}, 1×10^{-5}), different network structures, and ϵ values (0.02, 0.05, 0.10).

In order to help better understand the conjoint optimization mechanism of DQN routing and facilitate tuning of the model, DQN models with different parameters are trained and then applied to solve a number of parameterized routing problems generated with problem sets generator. Table 2 lists the parameters of generated problems and DQN model parameters in the experiments. In the table, *net numbers, capacity (global), and max pin numbers (each net)* belong to problem parameters; while *episodes, max step, γ, ϵ, and burn-in* belong to DQN model parameters. In order to demonstrate statistically significant results, each case is run on 40 problems, with the location of the pins and the number of pins randomly selected. In order to simulate reduced capacity specifications, the capacity for three of the most congested edges evaluated based on A* solutions is reduced.

5 Results and Discussion

5.1 Conjoint Optimization of Deep Q-Network Routing. The main advantage of the Q-network-based DRL for routing is the model's ability to consider subtasks (nets and pins to be routed) conjointly, which in this paper is referred to as *conjoint optimization* of the DQN model. In exploring the advantage of such a mechanism, a study of routing problem types is conducted beforehand.

Based on a series of experiments on the generated problems with different parameters, one key difference between the problems is based on how the edges are utilized in A* routing. In the first category, no edge that originally has a positive capacity is fully utilized based on A* and this category of problems is referred to as a *type I problem* or *no-edge-depletion problem*. In the second category, at least some edges that originally have a positive capacity are fully utilized based on A*, and this category is referred to as a *type II problem* or *partial-edge-depletion problem*.

Type I and type II problems' conditions are simulated by experiment parameter no. 5 (type II) and no. 8 (type I), respectively. The

Table 2 Problem and model parameters of the DRL routing experiment

No.	Net number	Capacity	Max pin number	Episodes	Max step	γ	ϵ	Burn-in	DRL win (%)
1	50	5	2	5000	50	0.98	0.05	A*	77.5
2	50	5	2	5000	50	0.95	0.05	A*	90
3	50	5	2	5000	50	0.9	0.05	A*	97.5
4	50	5	2	5000	50	0.8	0.05	A*	95
5	50	5	2	5000	50	1	0.05	A*	80
6	50	5	2	5000	50	1	0.05	Random	57.5
7	50	5	2	5000	50	1	0.05–0.01	A*	77.5
8	30	5	2	5000	50	1	0.05	A*	22.5

only different parameter for the two sets of problems is the net number: in experiment no. 5, there are 50 nets in each problem and in experiment no. 8, there are 30 nets in each problem. Under this parameter setting, type II problems appear more often in experiment no. 5 then they do in no. 8. Figure 6 shows the analysis of the DQN models performance on two types of problems as well as the plots showing edge utilization conditions of the two types of problems based on A* solutions. Figures 6(*a*) and 6(*e*) show the edge utilization plots of edges before and after A* for a type I (Fig. 6(*a*)) and a type II (Fig. 6(*e*)) problem. For type I problems, with fewer nets to be routed, no edges are fully utilized after A*; while for type II problems, more nets need to be routed within the same space. As such, some edges are fully utilized after A*, as indicated by the arrow in Fig. 6(*e*).

Figures 6(*b*) and 6(*c*) show the comparison of WL and OF based on DQN routing solutions and A* on 40 cases of no. 8 experiments (type I dominant); while Figs. 6(*f*) and 6(*g*) show the comparison of WL and OF based on DQN solutions and A* on 40 cases of no. 5 experiments (type II dominant). By comparing the performance of DQN routing solutions on the two types of problems, it can be seen that the DQN router outperforms A* in most cases (80% in terms of WL) in type II dominant cases, while in Type I dominant cases DQN router only outperforms A* in less than one-third of the cases (22.5% in terms of WL). In terms of OF, both DQN and A* solutions have no OF in type I dominant problems; while in type II dominant cases, DQN solutions still have no OF and A* has occasional OF. It is worth mentioning that DQN models have been hard coded to achieve zero OF in this study, while A* models cannot be forced to achieve zero OF since doing this may lead to A* failing to solve certain difficult problems.

The main reason for the difference in DQN router performance lies in the change of edge capacity when routing nets sequentially. To better illustrate this, some example capacity configurations are provided in Fig. 7, which shows illustrative capacity configurations of an original state (Figs. 7(*a*) and 7(*b*)), no-edge-depletion state (Figs. 7(*c*) and 7(*d*)), and partial-edge-depletion state (Figs. 7(*e*) and 7(*f*)). One can think of this as the capacity heat map in one of the three directions (horizontal direction, vertical direction, and the via direction) of a global routing problem. The no-edge-depletion state and partial-edge-depletion state are generated by occupying a proportion of the capacity of the original state, thus simulating the capacity condition during routing after a subset of the nets and a subset of the pins have been routed. Binarized plots are capacity plots thresholded at zero: grids with capacity equal to or below zero (corresponding to an overflow condition) are marked in black, while grids with positive capacities are marked in white.

For a type I (no-edge-depletion) problem, since no edge is totally utilized after routing all nets, when routing the nets and pin pairs sequentially, the capacity configuration will be similar to the case in Fig. 7(*c*) throughout the routing procedure. In this way, even though the capacity is changing after some of the nets and pins have been routed, the binarized capacity heat map would not change throughout the process. For A*, it only matters if a certain edge has a positive capacity (not-blocked) or zero capacity (blocked). In other words, A* finds solutions of a two-pin problem based on the binarized capacity heat map. With admissible heuristics, which is the case in this research, A* router is guaranteed to yield the optimum solution for each two-pin problem. Since the binarized capacity heat map would not change throughout the process in solving a type I problem, A* solutions are guaranteed

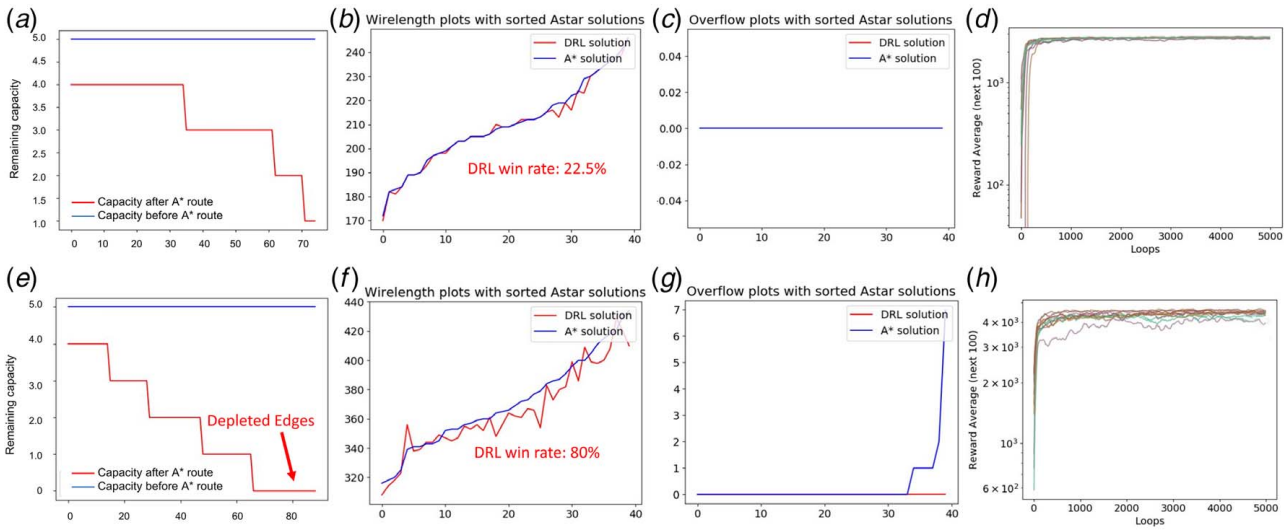

Fig. 6 Analysis on DQN model performance on two types of problems. Type I problem: no-edge-depletion type: (*a*) edge utilization plot; comparison of (*b*) WL and (c) OF based on DQN routing solution and A* solution, (*d*) log-plot reward curves of 10 cases. Type II problem: partial-edge-depletion type: (*e*) edge utilization plot; comparison of (*f*) WL and (*g*) OF based on DQN routing solution and A* routing solution, and (*h*) log-plot reward curves of 10 cases.

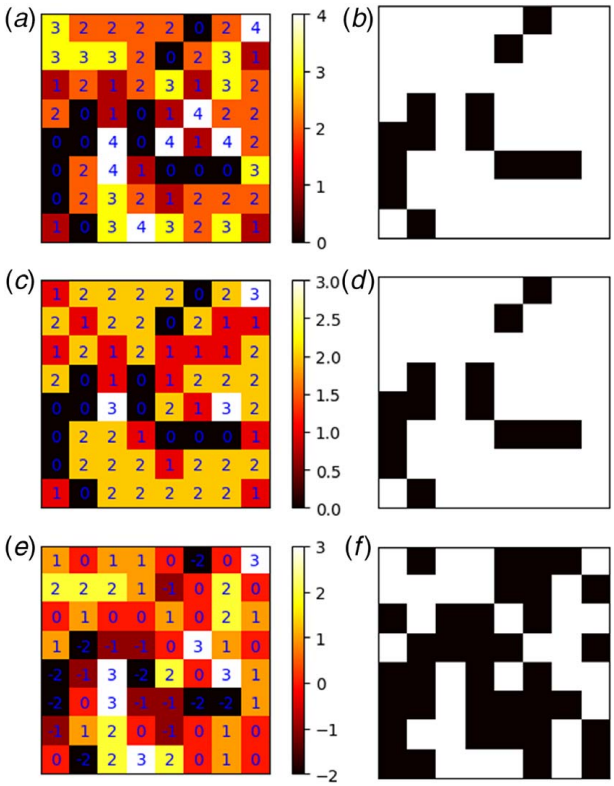

Fig. 7 Example capacity and their binarized plot. (*a*) and (*b*) original capacity; (c) and (*d*) no-edge-depletion capacity; (e) and (*f*) partial-edge-depletion capacity.

to provide the globally optimum solution for the entire routing problem given a pin decomposition algorithm for the following reasons: first, pre-routed nets will have no influence on the nets routed later, since the binarized capacity conditions are not changing; second, a routing problem solution is essentially a combination of solutions for each two-pin problem. Thus, in solving a type I problem, DQN router will not necessarily exhibit any advantage over A*, which can already find the optimum solution. The small proportion of DQN outperformed cases in which DQN router outperforms in no. 8 experiments (type I dominant) originate from the stochastic nature of the problem generator: even with fewer nets, there may occasionally be some type II problems generated.

However, when solving for a type II problem, some edges will be fully utilized at a certain stage of the routing process when a portion of the nets have been routed. Therefore, the partial-edge-depletion capacity condition similar to what happens in Figs. 7(*e*) and 7(*f*) will take place during the routing process. In a partial-edge-depletion scenario, A* will still solve each two-pin problems sequentially, without taking into consideration the change of the capacity conditions. In solving such problems, DQN-based router is guided by the reward and solving the problem in an iterative training process. Furthermore, the capacity information is also encoded into the state representation of the DQN model. In this way, DQN router is able to solve a routing problem conjointly taking into account all nets and pins that need to be routed, which is described as the conjoint optimization mechanism.

A comparison between the log-plot of reward curves when solving a type I problem (Fig. 6(*d*)) and a type II problem (Fig. 6(*h*)) provides indirect evidence on how the conjoint optimization mechanism of DQN works. In Fig. 6(*d*), which consists of ten randomly selected log-plot reward curves when solving type I problems, the reward increases significantly in the first few hundred episodes. After that, it increases slowly and reaches a plateau gradually. For the ten log-plot reward curves for the type II problems, although these curves have also ramped up significantly in the first few hundred episodes, they keep oscillating throughout the training process. This corresponds to the process of model weights adjustment in order to "reroute" existing solutions, which can yield better solutions by circumventing congested areas.

A more direct case study showing the conjoint optimization of the DQN router is based on the solution analysis and comparison of A* and DRL on a toy problem, consisting of ten nets (each one with a maximum of two pins) on an $8 \times 8 \times 2$ space. In solving this toy problem, DRL router with A*-based burn-in memory is used, γ is set to be 0.9 and ϵ is set to be 0.05. Figure 8 shows some of the results for the case study. Figures 8(*a*) and 8(*b*) are the heat maps showing edge utilization in the horizontal and vertical directions based on A* routing solutions. The via capacity heat map is not displayed here since in the setting of this work, via capacity is set to be high enough that no OF will happen. The exact solution of A* routing is shown in Fig. 8(*c*), which has WL of 94 and OF of 1. The solution for exactly the same problem based on DQN router is shown in Fig. 8(*d*), which has WL of 92 and OF of 0. By comparing the solutions of two routers, it is obvious that in the DQN router there are some reduced connections in both the vertical direction (in the purple and black circle) and the horizontal direction (in the purple circle). From the heat maps of A* routing, it can be seen that the reduced-connection areas correspond to edges that are more actively utilized, or in other words more congested. Such an optimization procedure is not hard-coded, but instead realized by the conjoint optimization of the DQN router.

The major reason why DQN conjointly optimizes the problem is that the DQN model solves a whole routing problem with a single model, thus taking into account all the subtasks (nets and pins to be connected in this work) simultaneously. For the A* router, each subtask is solved individually and in this way the results will be strongly influenced by the order in which the subtasks are solved. In effect, the pre-routed nets will strongly influence the quality of routing for pins and nets routed later. Although there can be heuristic-based algorithms to order the subtasks to combat this issue, such as routing nets with greater spanning areas first, such heuristics may not generalize well for all problems. For example, a spanning area-based net ordering heuristics will be highly ineffective when solving a routing problem in which the spanning areas of most nets are similar.

5.2 Deep Q-Network Model Fine Tuning. In order to obtain DQN models with enhanced performance, fine tuning of the model is conducted in two steps: *general parameter tuning* and

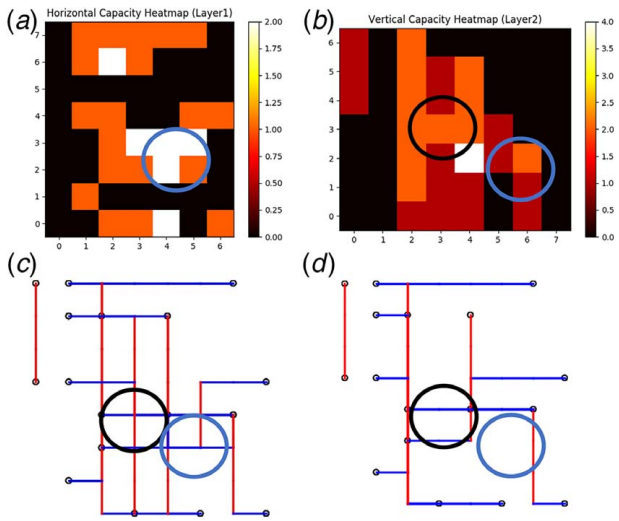

Fig. 8 Case study showing the conjoint optimization of DQN router: heat map showing capacity utilization on (*a*) horizontal direction and (*b*) vertical direction based on A* routing, (c) A* routing solution, and (*d*) DQN routing solution (layer 2; layer 1)

performance improvement tuning. General parameter tuning aims at stabilizing the training process and guarantee that there will be feasible solutions generated at the end of training, which includes: batch size, learning rate, network structures, and value of ϵ. After a general parameter tuning, a model performance tuning is conducted on further increasing the model's ability to yield more desirable routing solutions, which also consists of a decay ϵ strategy, γ. Finally, the choice of memory burn-in is studied and analyzed.

5.2.1 General Parameter Tuning. The performance of different models is evaluated in terms of reward evolution during DQN model training on $4 \times 4 \times 2$ toy problems with only 3 nets (each one with a maximum of 2 pins) and the training episodes is 2000. The simplified setting for general parameter tuning is based on our experimental results indicating the rough region of parameters that have a reasonable performance on most problems. The plotted rewards are not cumulative but the rewards summation for an episode, i.e., rewards summation for all two-pin routing sub-task in a problem. The rewards summation in an episode thus is obtained from the summation of rewards over all two-pin routing problems.

Batch size: Batch sizes of 32, 64, and 128 are studied and the reward plots of models across three different batch sizes are shown in Fig. 9. The results indicate that increased batch size does not cause the reward to increase faster and could lead to oscillations in the reward when the batch size is 128.

Learning rate: Experimented learning rates include 1×10^{-3}, 1×10^{-4}, and 1×10^{-5}. These produce a marked difference in the reward evolution during training as shown in Fig. 9. When the learning rate is small, the reward increase is slow and unstable. After 2000 episodes, the model fails to produce a reasonable solution. A more feasible learning rate typically already produces feasible solutions after 2000 episodes. On the other hand, when the learning rate is too large, strong oscillations exist throughout the whole training. A learning rate of 1×10^{-4} results in the best trade-off.

Network structures: Various parameters including the number of layers, the number of nodes each layer, and activation functions are studied. In particular, three fully connected networks with different number of nodes in their second layer are investigated. Figure 9 shows the reward plot of DRL models with three different networks. As shown, decreasing the node numbers of the second layer negatively influences the performance of DQN models by increasing the time it takes to obtain a higher reward or making the learning process unstable.

ϵ value: As shown in Eq. (8), the value of ϵ controls the exploration-exploitation rate with higher ϵ values corresponding to more randomized actions. Figure 9 shows the reward plots based on different ϵ values. When ϵ is 0.05, the reward evolution is more stable compared to cases in which ϵ is smaller or greater than 0.05. Besides, a commonly adopted decay ϵ strategy is experimented in the performance improvement tuning.

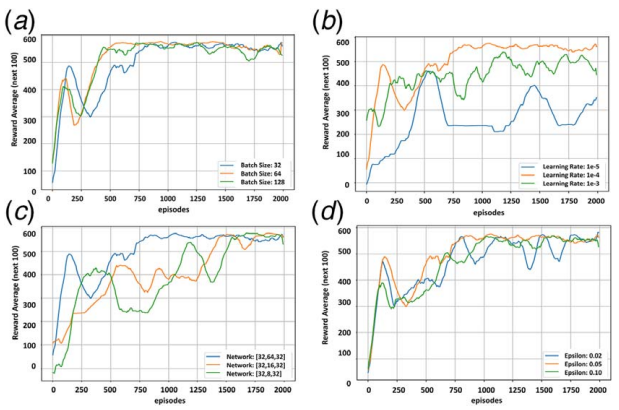

Fig. 9 Reward plots based on different combinations of model parameters: (*a*) batch size, (*b*) learning rate, (*c*) network structures, and (*d*) epsilon

Based on the above experimentation, our optimized parameters are as follows: batch size, 32; learning rate, 1×10^{-4}; nodes in layers, [32, 64, 32]; ϵ, 0.05. While the best parameter set may vary with problem complexity such as the grid size, the number of nets, and the number of pin pairs on a grid, our own experience indicates that the complexity of the network structure needs to be increased to accommodate the increase in problem complexity. However, a quantitative approach that reveals the best network structure as a function of the given model size without overfitting is the subject of our future studies.

5.2.2 Performance Improvement Tuning. Based on the previous general parameter tuning, a tuning aimed at improving the model performance for solving routing problems is conducted, focusing on γ and ϵ, which will significantly influence the model performance. First, a decay ϵ strategy, which linearly decreases ϵ from 0.05 to 0.01 starting from 2000 episodes, is experimented in experiment no. 7 shown in Table 2. Compared with experiment no. 5 which does not include decay ϵ strategy, there is a minor decrease in the performance. Therefore, the decay ϵ strategy is not adopted in the optimized models.

Then, models based on different values of γ (1, 0.95, 0.9, and 0.8), which is the discount factor determining the trade-off between short-term and long-term reward, are experimented to solve 40 problems with parameters specified in no. 2, no. 3, no. 4, and no. 5. In this research, short-term reward corresponds to the negative reward obtained from a random walk, while long-term reward corresponds to the positive reward obtained when the target pin is reached. Figure 10 shows the statistical results for routing solutions of DQN-based router with different values of γ. Figures 10(*a*)–10(*d*) show the WL for the 40 experimented problems of DQN routing solutions with γ values of 1, 0.95, 0.9, and 0.8 and their corresponding A* routing solutions. In most cases, DQN router gives better solutions with less WL and the percentage of problems in which DQN router yields better solutions are 80% ($\gamma = 1$), 90% ($\gamma = 0.95$), 97.5% ($\gamma = 0.9$), and 95% ($\gamma = 0.8$), respectively. Thus, the optimum γ based on the statistical results are chosen to be 0.9. Histograms in Figs. 10(*e*)–10(*h*) show the decrease in WL of DQN-based routing solutions on the 40 experimented problems compared to A*-based routing with γ values of 1, 0.95, 0.9, and 0.8. The one (Fig. 10(*g*)) that corresponds to best γ shows that on all the 40 problems experimented, WL gets non-negative decrease. It is worth mentioning that in these 40 problems, a small proportion of problems had a non-zero OF based on A* routing. However, in all cases, the OF of DQN routing solutions is zero. The above analysis indicates the overwhelmingly better performance of DQN router compared with A* router on type II problems.

The log-plot of reward curves for 10 randomly selected problems solved by DQN router with γ value of 1, 0.95, 0.9, and 0.8 is shown in Figs. 10(*i*)–10(*l*). The first observation is that oscillations of reward curves increased considerably as γ decreases, indicating greater adjustment of DQN-network weights to obtain conjointly optimized solutions. The reason for such a change is that when γ is too large and even close to one, the DQN model will no longer be sensitive to a change of WL represented by short-term negative reward. Only when γ becomes reasonably small and the short-term negative reward starts having a greater influence over the expected total reward, will the conjoint optimization start to work. Another important observation is that when γ becomes too small, especially when it is 0.8 (Fig. 10(*l*)), some reward curves plateau at a lower level compared to where they would have reached when γ is 1 (Fig. 10(*i*)). One plausible explanation for the degeneration of solutions as γ gets too small is that positive reward obtained from reaching a target pin diminishes when multiplied with multiple γ.

5.2.3 Burn-In Memory Choice. The burn-in memory for the DQN model is yet another key component and the choice of burn-in memory is regarded as a tunable parameter. Here, a comparison of a random versus A*-based burn-in memory, which use A* routing solutions for the same problem, is conducted to determine

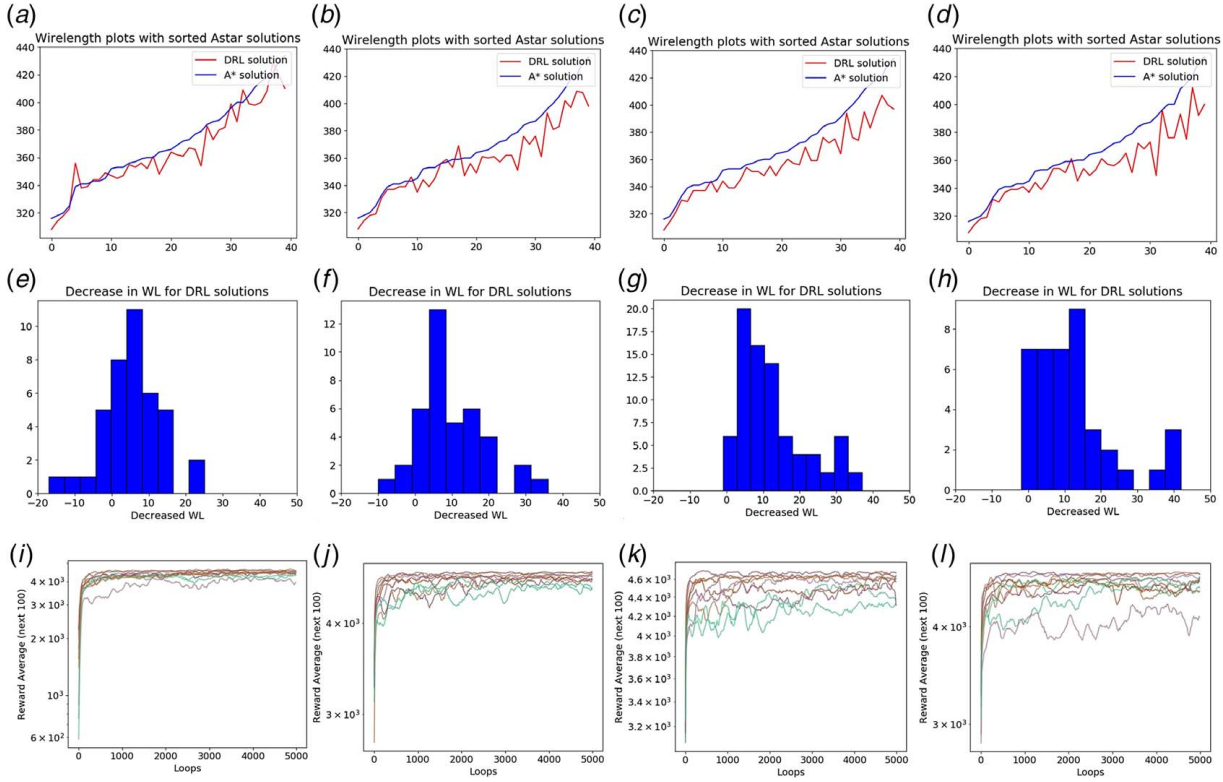

Fig. 10 Influence of discount factor γ on performance of DQN router and reward curves. WL of DQN routing solutions with γ value of (*a*) 1, (*b*) 0.95, (*c*) 0.9, and (*d*) 0.8 and their comparison to A* routing solutions; decrease in WL of DQN-based routing solutions compared to A* routing solutions with γ value of (*e*) 1, (*f*) 0.95, (*g*) 0.9, and (*h*) 0.8; log-plot of reward curves for ten randomly selected problems solved by DQN router with γ value of (*i*) 1, (*j*) 0.95, (*k*) 0.9, and (*l*) 0.8.

the appropriate choice of burn-in memory. Experimental results corresponding to no. 5 and no. 6 in Table 2 demonstrate the performance of DQN router on type II problems with A*-based and random memory burn-in, respectively. Figures 11(*a*) and 11(*c*) show the WL of DQN router with random and A* router memory burn-in, respectively, with the corresponding WL of A* routing solutions also plotted for comparison. For random memory burn-in case, DQN routing solutions show no signs of advantage and only in 57.5% cases have smaller WL compared with A* routing solutions. With the application of A* router memory burn-in, DQN routing solutions are mostly better than corresponding A* routing solutions. To be specific, in 80% of the problems experimented, DQN router can outperform A* router. In terms of OF, DQN routing solutions all have zero OF while A* router can occasionally have positive OF. For the log-plot of reward curves of ten randomly selected problems solved by DQN router with random (Fig. 11(*b*)) and A* router (Fig. 11(*d*)) based memory burn-in, the difference is minuscule. Thus, A* router memory burn-in is selected owing to its positive effect on performance improvement, yet it will not considerably increase the learning speed of models based on the experimental results of this research.

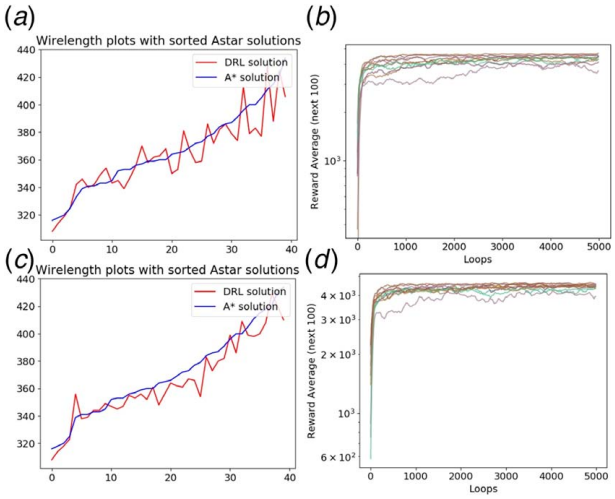

Fig. 11 Burn-in memory study showing statistical results of DQN router on type II problems: WL of DQN-based routing solutions with (*a*) random burn-in memory and (*c*) A*-based burn-in memory and their comparison to A*-based routing solutions; log-plot of reward curves for ten randomly selected problems solved by DQN-based routing with (*b*) random burn-in memory and (*d*) A*-based burn-in memory

5.3 Deep Reinforcement Learning Solutions on Different Problems. Based on the optimized network structure and model parameters, the visualized solutions of DQN routing solutions on both $8 \times 8 \times 2$ and $16 \times 16 \times 2$ problems that are both type II are presented in Fig. 12 with their respective WL and OF. The results show that the tuned DQN model not only yields better solutions on $8 \times 8 \times 2$ scale problems but also performs better when directly applied to solving $16 \times 16 \times 2$ scale problems, without change of model parameters or structures.

6 Conclusions

This work is the first attempt to formulate the historically challenging IC global routing as a deep reinforcement learning problem. By following the established conventions of solving problems sequentially and pin decomposition in global routing, the proposed DQN router produces superior solutions over an A* router in most cases for type II (partial-edge-depletion) problems, even though A* uses an admissible heuristic. The conjoint optimization

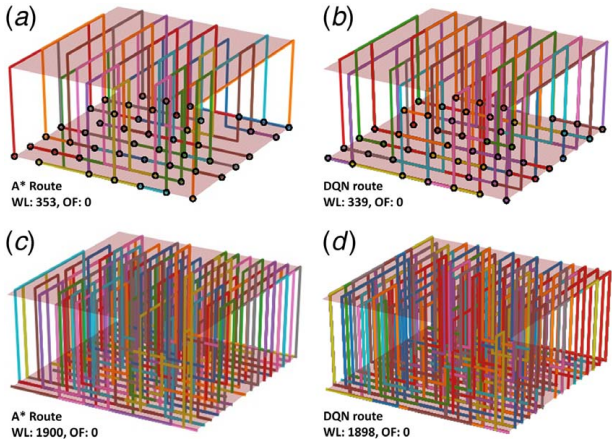

Fig. 12 Visualized results showing routing solutions for 8 × 8 × 2 problem based on (*a*) A* router and (*b*) DQN router; 16 × 16 × 2 problem based on (c) A* router and (*d*) DQN router

that leads to the superior performance of DQN router is analyzed and validated experimentally. Parameters including batch size, learning rate, network structures, ϵ, γ, and memory burn-in are fine-tuned to enhance the performance of the DQN model. An IC global routing problem sets generator is developed with the ability to generate global routing problem sets fast with user-defined size and constraints. These problem sets and code for the generator will be used for training our DRL approach and will be made publicly available to the community.

7 Future Work

There are two primary ways that the current work can be extended to make it applicable to bigger-sized problems. First, as pointed out in previous research, the conventional deep Q-network adopted in this work tends to overestimate the value of the Q-function. Better models such as double deep Q-learning (double DQN) [18] are promising alternatives for enhancing the performance. Second, only a single agent is used in our current approach, which does not naturally extend from global routing problems where each net often has more than two pins to be interconnected. Models with multiple agents [32–35] are likely to perform better at a set of simultaneous multi-pin problems, hence alleviating the restriction for solving the problem sequentially and therefore yielding better solutions. Also, alternative formulations and representation of global routing problems into DRL models such as formulating global routing as integer linear programming and solving with DRL are within the scope of our future work [36]. Graph convolutional neural networks [37,38] would be tried as a promising tool in reformulating existing global routing problems.

Finally, a major obstacle in applying advanced DRL or other machine learning-based algorithms to solve global routing problems is the absence of publicly available problem set repositories that include abundant global routing problems with varying sizes and constraints. In the follow-up work, we intend to train our model based on a large number of automatically generated problem sets that will not only help enable the scaling of the proposed system to larger grids with more complex net configurations but also increase the model's generalization ability to solve unseen problems. Prioritized experience replay approach [39] will be utilized to train the Q-network with important transitions more frequently.

Acknowledgment

This work is partially funded by the DARPA IDEA program (HR0011-18-3-0010; Funder ID: 10.13039/100006502). The authors would like to thank Rohan Jain, Jagmohan Singh, Elias Fallon, and David White from Cadence Design Systems for the constructive feedback and the establishment of the simulated routing environment.

Appendix A: Heat Maps Showing Edge Utilization of Different Problem Sets Based on A* Solution

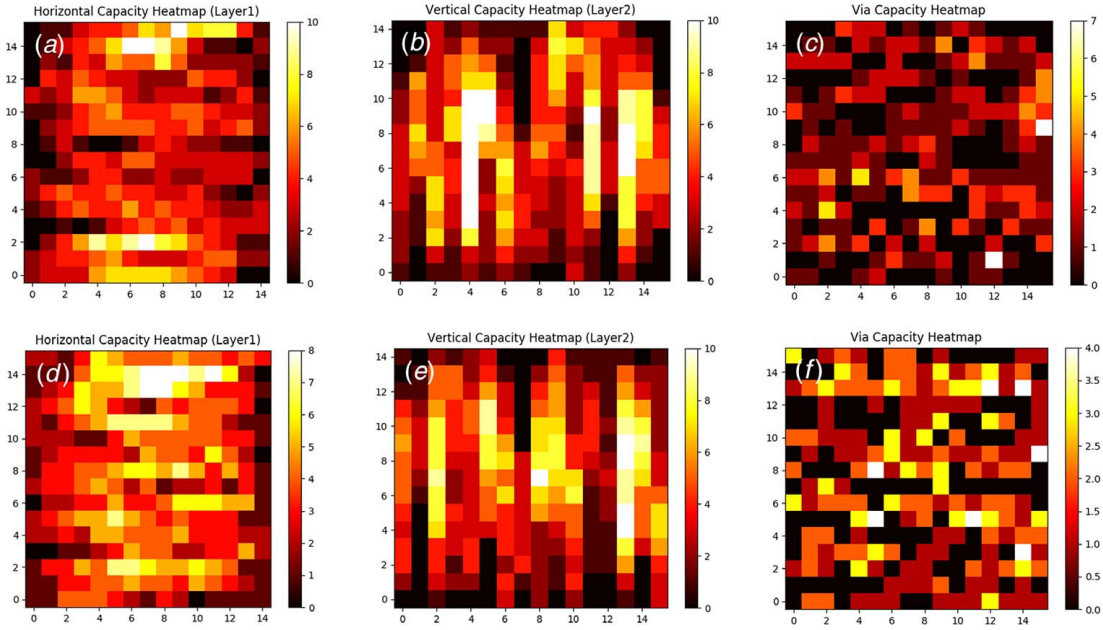

Fig. 13 Heat maps and edge utilization histogram of two global routing problems generated with the global routing problem sets generator. 16 × 16 × 2 benchmark with normal capacity setting: (*a*) via capacity heat map, (*b*) vertical capacity heat map, and (c) horizontal capacity heat map; 16 × 16 × 2 benchmark with reduced capacity setting: (*d*) via capacity heat map, (*e*) vertical capacity heat map, and (*f*) horizontal capacity heat map (number of nets, 160; max number of pins each nets, 2; normal capacity, 3).

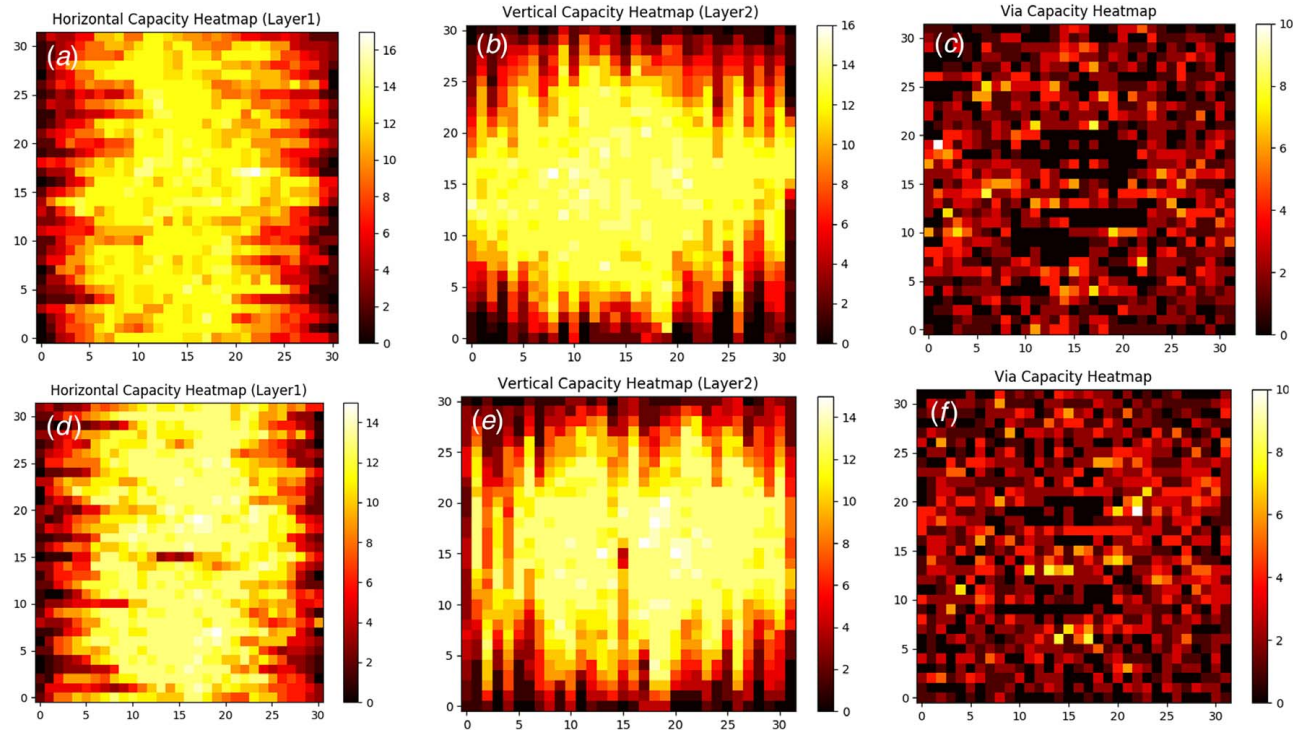

Fig. 14 Heat maps and edge utilization histogram of two global routing problems generated with the global routing problem sets generator. 32 × 32 × 2 benchmark with normal capacity setting: (*a*) via capacity heat map, (*b*) vertical capacity heat map, and (*c*) horizontal capacity heat map; 32 × 32 × 2 benchmark with reduced capacity setting: (*d*) via capacity heat map, (*e*) vertical capacity heat map, and (*f*) horizontal capacity heat map (number of nets, 160; max number of pins each nets, 2; normal capacity, 3).

Appendix B:

DQN Global Routing Algorithm

1. Burn in replay memory $D = e_1, e_2, \ldots, e_t$ to capacity N
$e_t = (s_t, a_t, r_t, s_{t+1}, IsTerminal_{t+1})$
2. Initialize action-value function Q with random weights
3. Set batch size: n
4. Decompose multi-pin problems with minimum spanning tree
5. Network training:

for episode = 1, maximum episodes
  Initialize sequence s_1 starting at one pin
  With probability ϵ, select a random action a_t
  otherwise select $a_t = \max_a Q^*(\phi(s_t, a; \theta))$
  Execute action a_t in environment and observe reward r_t
  Update the **environment** (capacity)
  for t = 1, maximum steps
    Set $s_{t+1} = s_t, a_t$
    Store transition $(s_t, a_t, r_t, s_{t+1}, IsTerminal_{t+1})$ in D
    Sample random minibatch of transitions from D

$$\text{Set } y_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta) & \text{for non} - \text{terminal } s_{j+1} \end{cases}$$

    Perform a gradient descent step on $(y_i - Q(s_j, a_j; \theta))^2$
  end for
end for

References

[1] Kahng, A. B., 2018, "Machine Learning Applications in Physical Design: Recent Results and Directions," Proceedings of the 2018 International Symposium on Physical Design, Seaside, CA, Mar. 25–28, ACM, pp. 68–73.
[2] Schaller, R. R., 1997, "Moore's Law: Past, Present and Future," IEEE Spectrum, **34**(6), pp. 52–59.
[3] Hu, J., and Sapatnekar, S. S., 2001, "A Survey on Multi-Net Global Routing for Integrated Circuits," Integration, **31**(1), pp. 1–49.
[4] Kramer, M. R., 1984, "The Complexity of Wirerouting and Finding Minimum Area Layouts for Arbitrary VLSI Circuits," Adv. Comput. Res., **2**, pp. 129–146.
[5] Chambon, R., and Tollenaere, M., 1991, "Automated Al-Based Mechanical Design of Hydraulic Manifold Blocks," Comput. Aided Des., **23**(3), pp. 213–222.
[6] Kang, S.-S., Myung, S., and Han, S., 1999, "A Design Expert System for Auto-Routing of Ship Pipes," J. Ship Prod., **15**(1), pp. 1–9.
[7] Christodoulou, S. E., and Ellinas, G., 2010, "Pipe Routing Through Ant Colony Optimization," J. Infrastruct. Syst., **16**(2), pp. 149–159.
[8] Grayman, W. M., Clark, R. M., and Males, R. M., 1988, "Modeling Distribution-System Water Quality; Dynamic Approach," J. Water Res. Plan. Manage., **114**(3), pp. 295–312.
[9] Ehmke, J. F., Steinert, A., and Mattfeld, D. C., 2012, "Advanced Routing for City Logistics Service Providers Based on Time-Dependent Travel Times," J. Comput. Sci., **3**(4), pp. 193–205.
[10] Barceló, J., Grzybowska, H., and Pardo, S., 2007, "Vehicle Routing and Scheduling Models, Simulation and City Logistics," Dynamic Fleet Management, Springer, New York, pp. 163–195.
[11] Lee, J., Bose, N., and Hwang, F., 1976, "Use of Steiner's Problem in Suboptimal Routing in Rectilinear Metric," IEEE Trans. Circuits Syst., **23**(7), pp. 470–476.
[12] Moffitt, M. D., 2008, "Maizerouter: Engineering an Effective Global Router," IEEE Trans. Comput. Aided Des. Integr. Circuits Syst., **27**(11), pp. 2017–2026.
[13] Kastner, R., Bozorgzadeh, E., Sarrafzadeh, M., and Sarrafzadeh, M., 2000, "Predictable Routing," Proceedings of the 2000 IEEE/ACM International Conference on Computer-Aided Design, San Jose, CA, Nov. 5–9, New York, pp. 110–114.
[14] Cho, M., and Pan, D. Z., 2007, "Boxrouter: A New Global Router Based on Box Expansion and Progressive ILP," IEEE Trans. Comput. Aided Des. Integr. Circuits Syst., **26**(12), pp. 2130–2143.
[15] Hasan, N., 1987, "A Force-Directed Global Router," PhD thesis, University of Illinois, Urbana-Champaign, IL.
[16] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M., 2013, "Playing Atari With Deep Reinforcement Learning," preprint arXiv:1312.5602.
[17] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., and Legg, S., 2015, "Human-Level Control Through Deep Reinforcement Learning," Nature, **518**(7540), p. 529.
[18] Van Hasselt, H., Guez, A., and Silver, D., 2016, *Deep Reinforcement Learning With Double Q-Learning*, Vol. 2, AAAI, Phoenix, AZ, p. 5.
[19] Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J., 2016, "Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction

and Intrinsic Motivation," Advances in Neural Information Processing Systems, pp. 3675–3683.
[20] Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., Van Hasselt, H., and Silver, D., 2018, "Distributed Prioritized Experience Replay," preprint arXiv:1803.00933.
[21] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D., 2018, "A General Reinforcement Learning Algorithm That Masters Chess, Shogi, and Go Through Self-Play," Science, **362**(6419), pp. 1140–1144.
[22] Bellman, R., 1957, "A Markovian Decision Process," Indiana Univ. Math. J., **6**, pp. 679–684.
[23] Sechen, C., 2012, *VLSI Placement and Global Routing Using Simulated Annealing*, Vol. 54, Springer Science & Business Media, New York.
[24] de Vincente, J., Lanchares, J., and Hermida, R., 1998, "RSR: A New Rectilinear Steiner Minimum Tree Approximation for FPGA Placement and Global Routing," Proceedings of the 24th Euromicro Conference, PISA, July 10–13, Vol. 1, New York, pp. 192–195.
[25] Ho, J.-M., Vijayan, G., and Wong, C.-K., 1990, "New Algorithms for the Rectilinear Steiner Tree Problem," IEEE Trans. Comput. Aided Des. Integr. Circuits Syst., **9**(2), pp. 185–193.
[26] Wong, C. C. M. S. C., 1990, "Global Routing Based on Steiner Min-Max Trees," IEEE Trans. Comput. Aided Des., **9**, pp. 1315–1325.
[27] Cho, M., Lu, K., Yuan, K., and Pan, D. Z., 2007, "Boxrouter 2.0: Architecture and Implementation of a Hybrid and Robust Global Router," Proceedings of the 2007 IEEE/ACM International Conference on Computer-Aided Design, San Jose, CA, Nov. 4–8, New York, pp. 503–508.
[28] Qi, Z., Cai, Y., and Zhou, Q., 2014, "Accurate Prediction of Detailed Routing Congestion Using Supervised Data Learning," 2014 IEEE 32nd International Conference on Computer Design (ICCD), Seoul, Korea, Oct. 19–22, New York, pp. 97–103.
[29] Zhou, Q., Wang, X., Qi, Z., Chen, Z., Zhou, Q., and Cai, Y., 2015, "An Accurate Detailed Routing Routability Prediction Model in Placement," 2015 6th Asia Symposium on Quality Electronic Design (ASQED), Penang, Malaysia, Aug. 3–5, IEEE, pp. 119–122.
[30] Sutton, R. S., and Barto, A. G., 1998, *Introduction to Reinforcement Learning*, Vol. 135, MIT Press, Cambridge.
[31] Nair, V., and Hinton, G. E., 2010, "Rectified Linear Units Improve Restricted Boltzmann Machines," Proceedings of the 27th International Conference on Machine Learning (ICML-10), Haifa, Israel, June 21–24, pp. 807–814.
[32] Nguyen, T. T., Nguyen, N. D., and Nahavandi, S., 2018, "Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions and Applications," CoRR, abs/1812.11794.
[33] Hu, J., and Wellman, M. P., 1998, "Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm," ICML, Madison, WI, July 24–27, pp. 242–250.
[34] Buşonic, L., Babuška, R., and De Schutter, B., 2008, "A Comprehensive Survey of Multiagent Reinforcement Learning," IEEE Trans. Syst. Man Cybern. C (Appl. Rev.), **38**(2), pp. 156–172.
[35] Tan, M., 1993, "Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents," Proceedings of the Tenth International Conference on Machine Learning, Amherst, MA, July 27–29, pp. 330–337.
[36] Tang, Y., Agrawal, S., and Faenza, Y., 2019, "Reinforcement Learning for Integer Programming: Learning to Cut," preprint arXiv:1906.04859.
[37] Li, Z., Chen, Q., and Koltun, V., 2018, "Combinatorial Optimization With Graph Convolutional Networks and Guided Tree Search," Advances in Neural Information Processing Systems, pp. 539–548.
[38] Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A., 2019, "Exact Combinatorial Optimization With Graph Convolutional Neural Networks," preprint arXiv:1906.01629.
[39] Schaul, T., Quan, J., Antonoglou, I., and Silver, D., 2015, "Prioritized Experience Replay," preprint arXiv:1511.05952.