Circuit Routing Using Monte Carlo Tree Search and Deep Reinforcement Learning

Youbiao He
Dept. of Computer Science
Iowa State University
Ames, Iowa 50014
yh54@iastate.edu

Hebi Li
Dept. of Computer Science
Iowa State University
Ames, Iowa 50014
hebi@iastate.edu

Tian Jin
Dept. of Computer Science
Iowa State University
Ames, Iowa 50014
jtian@iastate.edu

Forrest Sheng Bao
Dept. of Computer Science
Iowa State University
Ames, Iowa 50014
fsb@iastate.edu

2022 International Symposium on VLSI Design, Automation and Test (VLSI-DAT) | 978-1-6654-0921-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/VLSI-DAT54769.2022.9768074

***Abstract*—We propose a new approach to circuit routing by modeling it as a sequential decision problem and solving it in MCTS with DRL-guided rollout. Compared with conventional routing algorithms that are either manually designed with domain knowledge or tailored to specific design rules, our approach can be reconfigured for nearly any routing constraints and goals without changing the algorithm itself because the AI agent explores solutions in a general search strategy. Experimental results on both randomly generated circuits and popular open-source hardware projects show that our method achieves 33.3% higher success rate than traditional A*-based approach.**

***Index Terms*—Circuit routing, Deep reinforcement learning, Monte Carlo tree search**

I. Introduction

In printed circuit boards (PCBs) and integrated circuits (ICs) design, routing is a major step to place metal wires to achieve expected connectivity of pins of all nets without violating design rules [1]. A correct layout of such wires is vital to the functionality and performance of the circuit. Circuit routing is computationally expensive and challenging. A PCB, such as the motherboard of a smartphone, can easily contain thousands of pins and ten layers, making manual design extremely time-consuming [2].

Due to the complexity of circuit routing, existing approaches typically divide it into two stages: first global routing and then detailed routing [3]. Global routing routes on large circuit blocks, called G-cells, with relaxations on many constraints, while detailed routing generates wires of proper shapes and positions based on results from global routing. However, the two stages do not always couple well. For example, it is difficult for the wire density in global routing results to match the routability of the downstream detailed router [4]. The miscoupling of the two stages becomes more profound as design rules evolve rapidly [5]. The distinction between the two stages also results in complex Electronics Design Automation (EDA) software that is difficult to maintain and evolve coherently.

In addition, routing algorithms are dominantly manually crafted based on domain knowledge. Although laborious, manual updates are always needed when new design constraints and goals arise.

To address the above problems, we propose an end-to-end and general routing approach that combines Monte Carlo tree search (MCTS) with a policy learned via deep reinforcement learning (DRL). As a powerful tool for the sequential decision, MCTS [6] takes discrete actions, one at each time, to build a path from the initial state to the terminal, and repeats such path generation for multiple iterations with an efficiency much higher than brute force to explore different solutions. The more it iterates, the closer it is to the optimal solution. It has been widely used to achieve computer's human or superhuman level play of games such as Go [7], chess, and shogi [8].

Inspired by its success, we view circuit routing as a sequential decision process that expands a path by adding a vertex (or edge) onto a grid graph in each step, resembling placing a stone in the game Go, which is suitable to be solved by MCTS [6]. However, the search space of routing is too large for vanilla MCTS that uses random rollout. To better guide the search process, we use DRL to train a policy and combine it with a backtracking algorithm to guide the rollout of MCTS, and propose a path pruning mechanism to reduce the wirelength further.

Unlike the two-stage approaches [3], [4], our DRL-MCTS routing approach is end-to-end, free from the coupling problem. Furthermore, it does not require domain knowledge nor specialized heuristics. It can easily adapt to different design constraints and goals without changing the algorithm itself. Hence, it has lower implementation and maintenance costs, and the computer can improve its skills as it routes more and more circuits. Experiments on both randomly generated circuit layouts and popular open-source hardware projects show the effectiveness of the proposed algorithm. The major contributions of this paper are listed as follows:

- An end-to-end and general approach to circuit routing, free from the coupling problem in the conventional two-stage routing paradigm, reconfigurable for nearly any routing requirements without changing the algorithm itself, independent from human domain knowledge.
- Backtracking and DRL-guided rollout for MCTS.
- A path pruning mechanism for reducing wirelength.
- Experiments on randomly generated and real-world circuits demonstrate our approach's superiority over the A*-based method, vanilla MCTS, and DRL in routing.

II. Problem Statement

Our circuit routing approach starts from constructing a graph $G = (V, E)$ via sampling vertices and edges on a uniform Cartesian grid. All the intersections on the grid form the set of vertices V. Pins of nets and obstacles (i.e., a non-routable region such as a drill hole or clearance/courtyard zone

978-1-6654-0921-6/22/$31.00 @2022 IEEE

of a pin) are at the intersections of the grid, hence also in V. For every vertex $v_1 \in V \backslash O$ and each of its 1-hop neighbors $v_2 \in V \backslash O$, an edge $e = \{v_1, v_2\}$ is added into E, where O is the set of obstacles.

A *routing problem* is, given a graph $G = (V, E)$ and a set of k nets $\mathcal{N} = \{N_1, \ldots, N_i, \ldots, N_k\}$ ($N_i \subseteq V$), to find a path P_i for each net $N_i \in \mathcal{N}$ such that $P_i \cap P_j = \varnothing$ for any two distinct paths P_i and P_j. A routing problem is *unsolvable*, if there exists no solution for any net. In this paper, each net contains only two pins because a multi-pin net can be decomposed into multiple two-pin nets easily using a rectilinear Steiner tree (RST) [9].

III. Methods

A. Overview

We model circuit routing as a sequential decision problem. For every net, a path is constructed from one of two net pins, expanded by one edge for each step until reaching the other pin. A special variable h, representing the head of the path under expansion, is tracked. At each step (Fig. 1), neighbors (excluding those already visited, obstacles, or pins in other nets) of the head h are evaluated, and one of them is picked as the new h from which the path expansion continues. Similarly, in the game Go, the AI player determines the best grid intersection to add a stone at each of its turns.

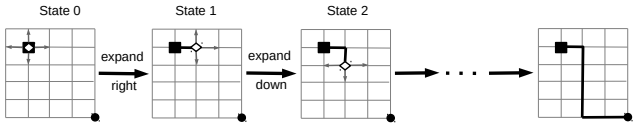


Fig. 1: Path expansion in 2D space from ■ to ●. ◇ indicates the head at each state. Action choices are arrows in gray. For multi-layer routing, just add two more actions to expand a path perpendicular to layers.

B. MCTS approach for routing

We formulate the routing problem as an MCTS problem [6] consisting of four following key components.

1) States: A *state* s is a collection of the graph G, the set of nets $\mathcal{N}$, the index of the net under connecting, the head h, the target pin, and all paths constructed so far.

2) Actions: At each state, the agent chooses an *action* a from the action set to expand the path to a direction. For example, four orthogonal actions in 2D space: up, down, left, and right (Fig. 5).

3) State transitions: Because the MCTS sequentially connects nets, two states differ only in the head, the target pin, and constructed paths. In this paper, the actions expanding the path to a non-routable region and the target pin are defined as *illegal action* and *connection action*, respectively. The state transition rules are:

(a) Initially, the head h is a randomly chosen pin of an unrouted net n, the target is the other pin of the net, and the path $P_n = \varnothing$.
(b) If the path is expanded to a node in the routable regions (except the target pin), then the head h is updated into the newly expanded node, and the path P_n is updated accordingly.
(c) If the agent takes an illegal action or a connection action from current state s_t, then the next state s_{t+1} is the *terminal state*.

4) Reward: In MCTS algorithms, reward is a function of the terminal state, which is determined by the path of current net in our problem. Therefore, we propose the following reward function.

$$r(P) = \begin{cases} -\alpha_{MT}|\mathcal{N}_u| - \beta_{MT} \sum\limits_{N \in \mathcal{N}_c} d_N - L_{MT}, & \text{if } P \text{ connects the net,} \\ -\mu, & \text{otherwise.} \end{cases} \tag{1}$$

where $\mathcal{N}_u$ is the set of unrouted nets who become unroutable due to the path P, $\mathcal{N}_c$ is the set of unrouted nets whose routability is not changed by the path P, d_N is the length difference of the shortest paths of net N before and after adding P to the circuit, α_{MT} and β_{MT} are the weights for each term (the suffix "MT" denotes the parameter is defined for MCTS), μ is the penalty for a failed routing, and L_{MT} is a user-defined loss for their other design rules and goals. Note that L_{MT} can be as simple as the total wirelength and could also cover other factors such as signal latency, EMC requirements, or signal length difference.

The first two terms of Eqn. (1) are regularization terms. They refrain the agent's goal from being over-optimized such that some nets become unroutable and/or the paths of some nets are stretched too long.

C. DRL-based policy training

In this paper, we use DRL to learn a policy to guide the search of MCTS. DRL policy cannot route the circuit directly because it only outputs an action distribution, the probability of successfully connecting a net by taking each candidate action. This is demonstrated by the experiments in Sect. IV-A. Therefore, a search mechanism such as MCTS is still needed to try on those actions to form a path.

Similar to an MCTS problem, an RL problem also has four key elements: *states*, *actions*, *state transitions*, and *reward* [10]. Note that in RL community, a state s is embedded into an *observation* o as the input to its policy function. States and actions of our DRL problem are similar to those in MCTS defined in Sect. III-B. The only change is that each action in DRL is embedded as a one-hot vector. State transitions and the reward are defined as follows:

1) State transitions: To steer the DRL agent to avoid creating paths that make unrouted nets unroutable, a *terminal state* is reached in DRL only when the DRL agent finishes connecting all nets (some of them can fail and get a penalty). In this way, the agent is guided to learn a policy that can successfully connect all nets to get the highest reward. DRL needs one more transition than MCTS, which is if the agent takes an illegal action or a connection action and the current net is not the last one to be routed, then switch to the next net in line and start from rule (a) in Sect. III-B3.

2) Reward: The *reward* is defined as a function of the current state and the chosen action:

$$R(s,a) = \begin{cases} \alpha_{RL}|\mathcal{N}_a| - \beta_{RL} \sum\limits_{N \in \mathcal{N}_b} z_N - L_{RL}, & \text{if } s' \text{ is a terminal state,} \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

where $\mathcal{N}_a$ is the set of nets that successfully connected, $\mathcal{N}_b$ is the set of nets whose path is expanded to a non-routable region, z_N is the Manhattan distance from the last node of the path of net N to the target pin, α_{RL} and β_{RL} are the weights for each term, L_{RL} is a user-defined reward function for their circuit design goals like L_{MT} in Eqn. (1), and s' is the next state of s after taking action a. The first two terms under the first condition aim at encouraging each path to be expanded to the target pin or a region close to the target pin. The DRL agent will get a higher value when more nets are successfully connected or the last nodes of paths are closer to the target pins.

In this paper, we adopt one of the most popular policy optimization approaches, Proximal Policy Optimization (PPO) [11], to update the policy. Compared to other policy-based methods such as REINFORCE [12], and TRPO [13], PPO is simpler to implement, converges faster, and has better sample complexity [11].

D. DRL-backtracking rollout

To further reduce the search space of MCTS, we use the DRL policy to guide a backtracking search as the rollout of MCTS. In vanilla MCTS, all rollouts are independent. Although two lengthy rollouts may share a great part of their paths, the expensive search on the shared part needs to be done twice. To improve search efficiency, we introduce backtracking into the rollout stage (Fig. 2). The rollout will not stop after reaching a failed terminal state. Instead, it backtracks to the previous state and chooses another action. In this way, the previous expensive steps will not be repeated and a connection is guaranteed if it exists.

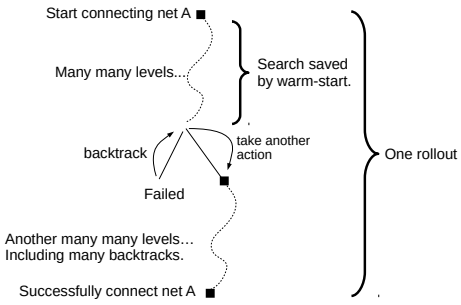


Fig. 2: Rollout with a backtracking mechanism.

Note that the output of the DRL policy contains illegal actions, which the MCTS agent should not take. To solve this problem, we change the probabilities of illegal actions to 0 and normalize the legal ones using their sum.

E. Path pruning

Although the DRL-backtracking mechanism significantly enhances the connection rate and reduces the search space of MCTS, a path can still contain some redundant parts, i.e., unnecessary detours to the targets. To prune paths, we examine each node pair on a path and replace the original path connecting the two nodes by a shorter and straight path if it exists and the area bounded by the original and shorter paths does not contain any pins of other nets. Note that the second condition is to check if the new path will change the way of other nets' connection compared with the original one. Fig. 3 shows an example of path pruning.

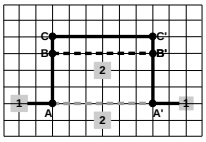


Fig. 3: An example of path pruning. The black solid line denotes the original path of net 1. The gray and black dashed lines are the shorter and straight paths to connect A and A', and B and B', respectively. The gray one cannot replace the original path because one pin of net 2 is in the region bounded by the closed path (A, C, C', A', A).

.

Our DRL-MCTS routing approach is summarized in Algorithm 1.

Algorithm 1: DRL-MCTS algorithm

```
Input: an initial circuit board B = (G, O, N, P), DRL policy π(a|s)
Hyperparameters: number of MCTS iterations T
for n ∈ [1..k] do
    Generate an initial state s_0 from B for net N_n.
    Create root node v_0 with s_0
    v ← v_0, s ← s_0.
    P_n = ∅.
    for t ∈ [1..T] do
        Update v and s by MCTS selection and expansion.
        Search a path P_t using π(a|s)-guided backtracking search from state s.
        Prune P_t.
        P_n ← P_n ∪ {P_t}
        Backpropogate r(P_t).
    end
    P ← P ∪ {arg max_{P∈P_n} r(P)}
end
Return P.
```

IV. Experimental Results and Analysis

In this section, our proposed method is examined on both randomly generated routing problems and real-world circuits. We further show that our method can accommodate different routing scenarios without changing the algorithm itself.

A. Our approach vs. baselines

Our method is a general framework that can be applied to a great variety of circuit routing needs. To help analyze

the result, we pick a simple but representative scenario to compare it with baselines. In this scenario, the circuit is single-layer, only 90-degree bends are allowed (i.e., four orthogonal actions: up, down, left, and right), and the only design goal is to minimize the total wirelength. To minimize the total length, the user-defined rewards for MCTS and DRL are defined as $L_{MT} = 0.1L_P$ and $L_{RL} = 0.1L_{\mathcal{P}}$, respectively, where L_P and $L_{\mathcal{P}}$ are the lengths of a path P and all paths routed in $\mathcal{P}$, respectively. Accordingly, the observation o for DRL is defined as a 4-dimensional vector containing the position of the head (first two elements) and its horizontal and vertical distances to the target pin (last two elements). All the parameter settings for PPO and MCTS are listed in Table I.

TABLE I: List of hyperparameter values.

Parameter	Value	Parameter	Value
Hidden layers	2	Discount (γ)	0.99
Neurons for each layer	32	Batch size	50
Activation function	ReLU	GAE parameter (λ)	0.97
Learning rate	1×10^{-4}	Clip ratio	0.2
Entropy coefficient (e_c)	0.1	PPO epochs	3000
Samples per epoch	1000	Target KL	0.01
α_{MT}	10	β_{MT}	0.1
α_{RL}	20	β_{RL}	0.5
μ	100	Exploration constant (C)	5
MCTS iterations	500		

One hundred randomly generated, routable single-layer circuits on a grid of 30×30 are used as the testbed. The number of nets for each circuit is randomly generated, and the maximum is 10. Two kinds of obstacles are generated: rectangular chips of random sizes from 2×2 to 7×7 and random scatter obstacles of size 1×1. Each circuit has two to four chips placed at random locations. A non-chip vertex has a one in 15 chance to become a scatter obstacle. The location of each net pin is randomly chosen around chips.

The baselines are traditional sequential A* routing [14], routing based on the vanilla MCTS, and routing based on PPO. They share the same hyperparameter values with our approach listed in Table I.

Successfully routing all circuits, our approach outperforms all baselines (Table II). Sequential A* solves 75 circuits or 75%. It could make some nets unroutable by pursuing the shortest path for the current net. PPO and vanilla MCTS fail to route any circuit before timed out. A DRL policy only gives a probability for each action to successfully lead the path to the target but does not guarantee that. The search space is too large for MCTS to find a solution within an applicable number of iterations.

Table II also reports the wirelengths of our approach and sequential A*. Because sequential A* can solve 75% circuits, the wirelengths in the table are the average values for the 75 circuits that both our approach and sequential A* can solve. Our approach generates paths that are 0.5% longer than that of sequential A*. The DRL and MCTS agents can learn a different but longer path from the A* in some cases to make it more possible for other nets to be connected later.

TABLE II: Success rate (%) and average wirelength (grid units)

	Our approach	Sequential A*	Vanilla MCTS	PPO
Success rate	100	75	0	0
Wirelength	140.67	139.95	-	-

B. On real-world circuits

The most exciting part of the experiments is testing our approach on real-world circuits. This study selects the top 30 KiCAD (a de facto PCB design suite for open-source hardware projects) projects on GitHub ranked by the number of stars. The routing problems for these projects are formulated according to our problem settings in Sect. II. Specifically, for each PCB file, we randomly select either the top or bottom layer to make a one-layer PCB and convert it into a grid with a resolution of 0.5mm to have an appropriate density of circuit components, e.g., the density in Fig 4. For each grid, we crop it at the center with the size of 32×32 to generate a test circuit. Problems that are not solvable in a single layer are excluded, resulting in 14 problems left with guaranteed routability.

All real-world circuits can be successfully routed by both our approach and the sequential A*. Our approach outperforms the sequential A* with a 102.7 versus 106.7 grid units of wirelength because paths of some connected nets stretch the paths of unrouted ones while using the sequential A* approach. Fig. 4 shows the routing results for one of them. In this example, the circuit contains 15 nets, and all nets are connected with a total wirelength of 236.

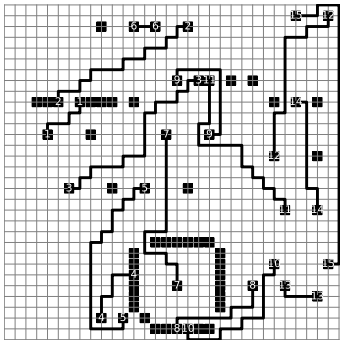


Fig. 4: Routing result of a simplified real-world circuit.

C. Flexibility analysis

As mentioned earlier, one advantage of our approach over existing solutions is that the algorithm is reconfigurable for nearly any routing requirements without changing the algorithm itself. To meet a new design need, only the problem formulation parameters, such as the reward function or the actions, need to be changed. Here we will use one example to show how to reconfigure the routing needs and achieve different results on the 100 generated test circuits.

Up to this point, in this paper, bends are 90-degree. Relaxing the bends to 45 degrees can often result in shorter paths

than those with 90-degree bends. To reconfigure for 45-degree bends, we just need to add four diagonal actions on top of the four orthogonal ones, and embed them using three legal directions (Fig. 5). This change requires the observation definition to be revised as well. The new observation is defined as a 6-dimensional vector, the first four of which are the same as that in the 90-degree bend case. The fifth and sixth elements are the x, y coordinates of the previous node on the path. Accordingly, the number of hidden layers and neurons for each layer of the policy network is changed to 4 and 64, respectively. Other settings are the same as those of the 90-degree bend case.

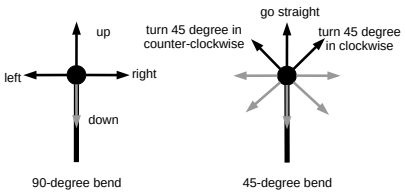
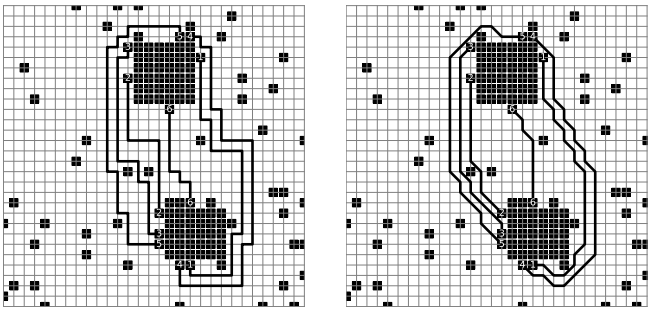


Fig. 5: Legal (black arrows) and illegal actions (gray arrows) at the head (●) of an under-construction path (black solid line).

In this case, the 100% success rate can still be achieved within 500 MCTS iterations. The average wirelength is 107.37 grid units for all the test circuits, shorter than that (141.52 for all) with 90-degree bends. Fig. 6 shows an example routing of our approach on one randomly generated circuit in different scenarios. It successfully connects all nets and the wirelength in 45-degree bend case is 24% shorter than that in 90-degree bend case for this circuit. Therefore, our approach can cope with the change on the bend angle by changing the settings of observations, states, actions, and state transitions.



(a) In 90-degree bends. (b) In 45-degree bends.

Fig. 6: Routing results of our approach on a generated circuit.

Based on the results above, our approach can solve circuit routing problems that have different constraints. To deal with an unseen problem with new constraints, only the DRL problem settings and hyperparameters, such as observations, actions, and network structure, need to be changed while the algorithm remains the same.

V. Conclusions

In this paper, we propose an end-to-end solution to circuit routing by combining deep reinforcement learning (DRL) and Monte Carlo tree search (MCTS). Specifically, we develop a DRL-based backtracking mechanism to reduce the research space and a path pruning method to reduce the wirelength. This approach can be easily implemented and adapted to nearly any design constraints and goals without changing the algorithm itself. Experimental results show that our approach can solve problems that baselines cannot.

References

[1] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, *VLSI physical design: from graph partitioning to timing closure*. Springer Science & Business Media, 2011.

[2] Hui Kong, Tan Yan, and M. D. F. Wong, "Automatic bus planner for dense PCBs," in *2009 46th ACM/IEEE Design Automation Conference*, July 2009, pp. 326–331.

[3] H.-Y. Chen and Y.-W. Chang, "Chapter 12 - global and detailed routing," in *Electronic Design Automation*, L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, Eds. Boston: Morgan Kaufmann, 2009, pp. 687 – 749.

[4] Y. Zhang and C. Chu, "Gdrouter: Interleaved global routing and detailed routing for ultimate routability," in *DAC Design Automation Conference 2012*, 2012, pp. 597–602.

[5] D. Shi and A. Davoodi, "Trapl: Track planning of local congestion for global routing," in *Proceedings of the 54th Annual Design Automation Conference 2017*, ser. DAC '17. New York, NY, USA: ACM, 2017, pp. 19:1–19:6.

[6] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, March 2012.

[7] D. Silver, A. Huang, C. J. Maddison, A. Guez *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[8] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[9] J. Hu and S. S. Sapatnekar, "A survey on multi-net global routing for integrated circuits," *Integration*, vol. 31, no. 1, pp. 1 – 49, 2001.

[10] A. Mirhoseini, A. Goldie, M. Yazgan, J. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, S. Bae *et al.*, "Chip placement with deep reinforcement learning," *arXiv preprint arXiv:2004.10746*, 2020.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[12] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[13] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul. 2015, pp. 1889–1897.

[14] H. Liao, W. Zhang, X. Dong, B. Poczos, K. Shimada, and L. Burak Kara, "A Deep Reinforcement Learning Approach for Global Routing," *Journal of Mechanical Design*, vol. 142, no. 6, Nov. 2019.