Asynchronous Reinforcement Learning Framework for Net Order Exploration in Detailed Routing

Tong Qu*†, Yibo Lin‡, Zongqing Lu‡, Yajuan Su*†§, Yayi Wei*†§
*Institute of Microelectronics of the Chinese Academy of Sciences, Beijing, China
†University of Chinese Academy of Sciences, Beijing, China
‡CS Department, Peking University, Beijing, China
§Guangdong Greater Bay Area Applied Research Institute of Integrated Circuit and Systems, Guangdong, China

2021 Design, Automation & Test in Europe Conference & Exhibition (DATE) | 978-3-9819263-5-4/21/$31.00 ©2021110.23919/DATE51398.2021.9474007

***Abstract*—The net orders in detailed routing are crucial to routing closure, especially in most modern routers following the sequential routing manner with the rip-up and reroute scheme. In advanced technology nodes, detailed routing has to deal with complicated design rules and large problem sizes, making its performance more sensitive to the order of nets to be routed. In literature, the net orders are mostly determined by simple heuristic rules tuned for specific benchmarks. In this work, we propose an asynchronous reinforcement learning (RL) framework to search for optimal ordering strategies automatically. By asynchronous querying the router and training the RL agents, we can generate high-performance routing sequences to achieve better solution quality.**

I. Introduction

Routing is a critical and time-consuming step in physical design [1]. Its solution impacts timing, power, and yield [2]. Routing is usually divided into global routing and detailed routing, with the former planning the rough routing regions and the latter finishing the actual interconnections [3]. Unlike global routing, detailed routing needs to handle plenty of design rules on a large grid graph. With feature sizes scaling down with the technology nodes, the routing grids become increasingly denser, leading to more complicated design rules from manufacturing, such as parallel-run spacing, end-of-line spacing, corner-to-corner spacing, and minimum area [4], [5]. Meanwhile, the grid graph for detailed routing is much larger than that of global routing, indicating larger solution space. As a result, detailed routing is becoming the most time-consuming step in advanced technology nodes [4].

While routing has been studied for several decades with many fundamental algorithms proposed, e.g., Lee's algorithm, A* search, negotiation-based rip-up and reroute scheme, etc., most of the attention has been paid to global routing for a long time [6], [7], [3]. In the past few years, with advanced technology nodes coming to the stage, the importance of detailed routing has been realized. Various aspects of detailed routing have been investigated. For example, pin access issues have been discussed in [8], [9], [10]. Ahrens et al. explored specific data structures for efficient detailed routing [11]. Manufacturing constraints have also been explored in [12], [13], [14], such as multiple patterning lithography friendly routing algorithms.

In recent ISPD contests [4], [5], detailed routing has been raised as a fundamental challenge in the backend design with practical benchmarks and realistic design rules. The contests largely stimulate the researches in detailed routing and several high-performance and robust routers have been proposed [15], [16], [17], [18], [19]. Sun et al. [20] proposed a valid pin access pattern generalization with a via-aware track assignment to minimize the overlaps between the wire segments. TritonRoute [15] adopted integer linear programming (ILP) for parallel intra-layer routing. DRAPS [18] developed an A*-interval-based path search algorithm to handle complicated design rules. Dr.CU [16], [17] proposed an optimal correct-by-construction path search algorithm and a two-level sparse data structure for runtime and memory efficiency. RDTA [19] developed an analytical approach to solve the track assignment problem following the global routing guides. Attention router explored reinforcement learning to solve the analog routing problem at a small scale [21].

Among the aforementioned detailed routers, most of them substantially follow the sequential routing strategy with the negotiation-based rip-up and reroute scheme [20], [18], [16], [17].

The parallelism is usually obtained by routing a batch of nets far away enough from each other simultaneously. This means the routing order of nets is critical to the performance of the algorithm. Currently, the net ordering strategy is usually determined by simple heuristics, e.g., the region size covered by a net. The performance may vary from design to design and from router to router as well. Therefore, a generic way to search for a good ordering strategy is desired to achieve high-performance routing.

To find a good ordering strategy, in this work, we formulate the strategy search problem into a reinforcement learning (RL) task to automatically learn from the designs. The major contribution can be summarized as follows.

- We formulate the ordering strategy search problem in sequential detailed routing into a reinforcement learning task such that it can be automatically learned.
- We develop an asynchronous reinforcement learning framework to learn from multiple designs simultaneously. By developing a customized neural network architecture, we can apply the learned model to different designs as well.
- Experimental results on ISPD 2018 & 2019 contest benchmarks [4], [5] demonstrate that the ordering strategy obtained from our framework generalizes well and can achieve 14% fewer DRC violations and 0.7% less total costs compared with the state-of-the-art detailed router Dr.CU 2.0 [17].

The rest of this paper is organized as follows. Section II explains the background of routing, reinforcement learning, and problem formulation. Section III presents the algorithm details. Section IV reports the experimental results on ISPD contest benchmarks. Finally, Section V concludes the paper.

II. Preliminaries

In this section, we introduce the background on VLSI routing, reinforcement learning, and problem formulation.

The first two authors contributed equally to this work. Yajuan Su, Yayi Wei are the corresponding authors (e-mail:{suyajuan, weiyayi}@ime.ac.cn).

A. Design Rules

More design rules are introduced in the advanced technology nodes. Meanwhile, three fundamental and representative design rules need to be considered [4]. (1) Short: a via or wire segment of a net should not overlap with any object of another net. (2) Spacing: the spacing between two objects should satisfy the minimum distances. There are several different types of such requirements, e.g., end-of-line spacing, parallel-run spacing, and cut spacing. (3) Minimum area: a metal polygon should have an area larger than the minimum threshold. Typical objectives for routing are to minimize the total wirelength and the DRC violations.

B. Dr.CU 2.0 Sequential Detailed Router

In year 2018 and 2019, the ISPD contest was organized on detailed routing [4], [5]. Dr.CU [17] won the first place in the ISPD 2019 contest and is open source. In this work, we adopt Dr.CU as the target detailed routing framework for studying, while the methodology can work on other routers as well. Fig. 2 illustrates its routing flow, which is a typical procedure for most sequential routing algorithms as well. Given a placed netlist, routing guides, routing tracks, and design rules, it first assigns access points for each pin. Then it starts the rip-up and reroute (RRR) iterations to accomplish the routing. During the RRR iterations, if the router encounters congestion or DRC violations when trying to route a net, it rips up the net and the conflicted nets, leaving them for the next iteration to reroute. With enough iterations, the router can achieve convergence. Finally, it performs a post-routing refinement stage to reduce DRC violations. It needs to be noted that within each RRR iteration, Dr.CU also exploits parallelism between nets far away from each other, such that there will be no interaction when simultaneously solving the routing problem of each net. This does not change the sequential nature of the algorithm, i.e., routing in a net-by-net manner, as it does not determine the routing of different nets at the same time.

The solution quality of sequential routers like Dr.CU is highly correlated to the order of nets to be routed. Fig. 1 shows the distribution of solution quality with random net ordering routed by Dr.CU. Although the wirelength does not change much, the order affects both via count and the number of DRC violations. Thus, the ordering strategy needs to be carefully designed for high-quality routing across various benchmarks.

Dr.CU sorts nets by the routing region sizes (half-perimeter of the bounding box) of each net in descent order. In other words, nets covering large routing regions are routed first. However, we observe that the routing region sizes of different nets can be very similar, leading to random orders between these nets, and eventually causing high variations in the final violations. For example, Fig. 3 shows that 5293 nets have the same routing region size, accounting for 14.4 % of the total number of nets in benchmark `ispd18_test3`. Therefore, there is a potential to improve the routing performance by developing an ordering strategy considering more features.

C. Reinforcement Learning

Reinforcement learning enables the agent to learn a policy by interacting with the environment to maximize the cumulative reward. As illustrated in Fig. 4, at each step t, the agent observes a state s_t, takes an action a_t based on s_t, receives a reward r_t, and then the state stochastically transits to the next state s_{t+1}. The objective is to learn a policy $\pi(a|s)$ that maximizes the expected cumulative reward, starting from any state s.

In this work, we define the environment as the router, the agent as a net order planner that ranks the nets based on the features (state). The net ordering result is the action and the reward is positively related to the solution quality, e.g., total wirelength and DRC violations.

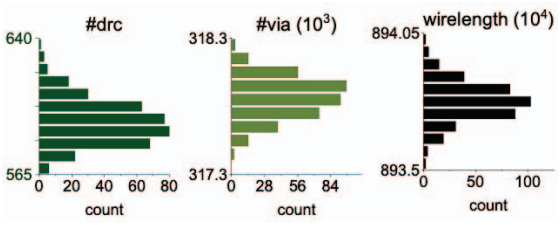

Fig. 1: Distribution of solution quality with random net ordering (300 iterations). The relative standard deviation of the number of DRC violations, the number of vias, and wirelength is 1.95 %, 0.04 %, and 0.008 %. Their ranges are 70.00, 845.00, and 4996.31, respectively.

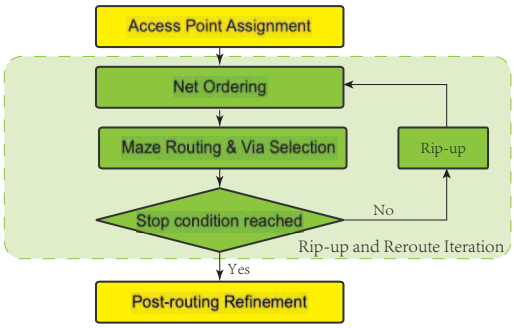

Fig. 2: Routing Flow.

D. Problem Formulation

We define the net ordering problem in detailed routing as follows.

Problem 1 (Net ordering)**.** *Given a set of nets N, train a net ordering*

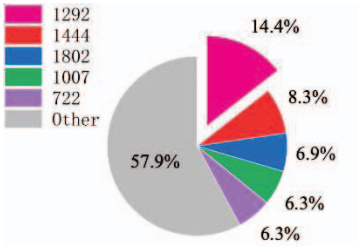

Fig. 3: Distribution of the net routing region sizes in `ispd18_test3`.

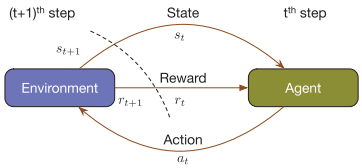

Fig. 4: Environment and agent system of reinforcement learning.

TABLE I: Features of Each Net.

Feature	Dimension	Description
Size	1	The size of the routing region (half-perimeter of bounding box).
Degree	1	Number of nets with conflicts in its routing region.
Count	1	The number of times it has been routed/rerouted.
Cost	1	The weighted sum of violations on it.
Via	1	Number of via on it.
WL	1	Wirelength.
LA	16	Layer assignment.

policy that can generate a ranking score s_i for each net $n_i \in N$ used by a sequential detailed router. The following metrics should be optimized simultaneously: (1) the total wirelength of all nets, (2) the number of the total used vias, (3) the number of DRC violations.

III. Algorithm

In this section, we first define the state space, action space, reward, and the basic RL setup. Then we explain the dedicated RL techniques for our routing problem. In the end, we summarize the overall flow of our RL algorithm.

A. Basic RL Setup

We define the state space, action space, and reward as follows:

State space $\mathcal{S}$: A state s is the collective representation of features for all nets. Table I summarizes the seven features for each net. The first feature is the size of its routing region. The second feature is its degree, which denotes the number of nets whose routing region overlaps with it. The third feature is the number of times routed/rerouted so far. The remaining four features are its costs information, including the violations cost, wirelength, number of vias, and metal layers assignment.

Action space $\mathcal{A}$: An action a is a real number vector. Each number is defined as an ordering score of a net.

Reward $\mathcal{R}$: Given the ordering scores (action a), the environment (router) will provide its feedback (i.e. evaluation metrics). The agent receives a reward according to the environment's feedback. The reward r is defined as:

$$r = -C_{\text{agent}} + C_{\text{cu}} \tag{1}$$

Where C_{agent} and C_{cu} are the total cost of all nets achieved by the agent's action a and Dr.CU's default strategy. The total cost C is defined as:

$$C = \sum_{i=1}^{4} w_i x_i \tag{2}$$

Where $x_i | i \in \{1, 2, 3, 4\}$ are the evaluation metrics used in the ISPD Contests, including short violation, spacing violation, number of vias, and wire length, $w_i | i \in \{1, 2, 3, 4\}$ are the weights of the above metrics. The objective of the agent is to learn a policy to maximize the reward.

B. The A3C Framework

Expensive query to the environment is a typical challenge in RL, leading to slow convergence and unaffordable training time. We adopt an asynchronous advantage actor-critic (A3C) method [22] with multiple actor-critic (AC) agents running in parallel. As shown in Fig. 5, each agent has a local copy of the policy and value networks. It performs actions in its environment to explore the solution space with a different policy. Different agents update the global network asynchronously during the training.

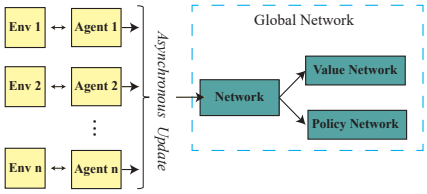

Fig. 5: The A3C Framework with asynchronous parallel agents and global network.

Algorithm 1 Update each A3C actor [22]

Require: Global shared parameter vectors θ, and θ_v.
1: Initialize thread step counter $t \leftarrow 1$
2: Define thread-specific copy of weights θ', θ'_v
3: **for** $T = 1, .., T_{max}$ **do**
4: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$ ▷ Reset gradients.
5: $\theta' = \theta$ and $\theta'_v = \theta_v$
6: Get state s_t
7: $t_{start} = t$
8: **repeat**
9: Find action a_t according to policy π
10: Sort nets according action a_t
11: Receive reward r_t and new state s_{t+1} from router
12: $t \leftarrow t + 1$
13: **until** terminal s_t or $t - t_{\text{start}} == t_{\max}$
14: Return $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \end{cases}$
15: **for** $i = t - 1, ..., t_{start}$ **do**
16: $R \leftarrow r_i + \gamma R$
17: $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i | s_i; \theta') (R - V(s_i; \theta'_v))$
18: $d\theta_v \leftarrow d\theta_v - \partial (R - V(s_i; \theta'_v))^2 / \partial \theta'_v$
19: **end for**
20: Perform async. update of θ using $d\theta$ and of θ_v using $d\theta_v$
21: **end for**

A3C maintains a policy $\pi(a_t | s_t; \theta)$ and an estimate of the value function $V(s_t; \theta_v)$, where θ and θ_v are the global shared parameter vector. The policy and the value function are updated after every $t_{\max}$ actions or when a terminal state is reached.

Algorithm 1 illustrates how each actor is updated. After initialization, each agent takes a copy of the global shared network, with parameters θ' and θ'_v (line 5), and then runs the policy for t_{max} steps or until a terminal state is reached. Finally, the agent computes the gradients in its process (line 17-18) and then updates the global share network asynchronously.

Network Architecture. We need two models in the A3C framework, a policy network and a value network. The policy network takes the state s and outputs two arrays (μ, σ^2) that represent a probability distribution over the actions. We pick the action by sampling from this probability distribution. We denote $\pi(a|s)$ as the probability of the sampled action a given state s. The value network outputs the value function $V(s)$ (the expected return in rewards for state s and action a), which is used to determine how advantageous it is in a particular state. Intuitively, the policy network tells us the ordering scores of the nets and the value network evaluates the scores in the sense of future rewards.

Fig. 6 plots the network architecture of the two models. We design the models in a special way so that the policy model can be used across different designs with different numbers of nets. To

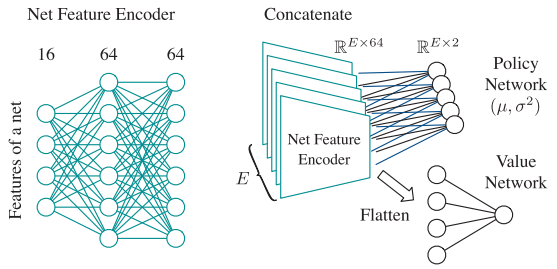

Fig. 6: Architecture of the policy and value networks, where E denotes the number of nets. The net feature encoder encodes the features of each net.

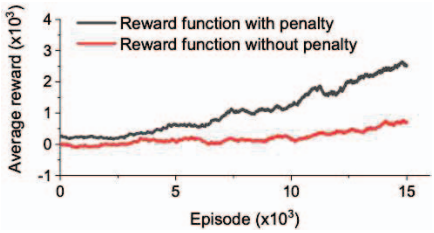

Fig. 7: Comparison of two reward functions. The curves represent the moving average reward of the last 1000 episodes (one episode includes four RRR iterations according to the setting of Dr.CU). We train the agents for 15 000 episodes and the maximum training time is within 24 hours. Mismatch penalty enables faster reward increase.

decouple the network architecture from the number of nets in design, we introduce a net-wise feature encoding network that encodes the features of each net independently. We then concatenate the encoded features for the policy and value networks. For example, given a design with E nets, the encoder will encode the $\mathbb{R}^{E\times 16}$ input feature tensor into an $\mathbb{R}^{E\times 64}$ tensor. The policy network takes this tensor and generates an array of ordering scores for all nets, i.e., $\mathbb{R}^{E\times 2}$ (mean and variance of the probability distribution for each net). We then sample from a normal distribution for each net to get its ordering score. In our implementation, μ is modeled by a linear layer and σ^2 by a softplus layer. The value network flattens the feature tensor and feeds into a fully connected layer with $E \times 64$ hidden units to obtain a scalar at the output.

The major benefit of such a network architecture is that the policy network can be shared across different designs, as we essentially perform net-wise modeling with the ordering score of each net dependent on its features only. While it is true that using a more complicated model that correlates the features of multiple nets may help to explore better policy, current architecture still has enough expressive power to verify the main idea of using RL in solving the net ordering problem. We leave the exploration of complicated models in the future.

C. Mismatch Penalty

General RL framework initializes the neural networks in a random manner, which may cause slow convergence in our problem, especially when obtaining the reward from the environment (i.e., running the router) is very time-consuming. On the other hand, we do have the prior knowledge to this problem that the default ordering strategy of using routing region sizes in Dr.CU is a generally good policy compared with a random one. Incorporating such knowledge has the potential to speed up training. Hence, Equation (1) is modified to:

$$r = -C_{\text{agent}} + C_{\text{cu}} - \frac{\alpha}{k}\sum_{i=1}^{k} {\Delta a_i}^2 \tag{3}$$

Where Δa is the difference between the predicted ordering scores and the sizes of routing regions, α is a user-defined parameter, and k is the number of nets to be routed. The parameter α is positive only at the early training steps and then set to zero. The detailed setup can be found in Section IV. Fig. 7 compares the learning speeds of the two reward function defining methods. The results show that the method of adding a mismatch penalty tends to learn faster. As we only apply the mismatch penalty at the early stage of the training, it will speed up the training, but not limit the exploration space to the heuristic ordering strategy used in Dr.CU.

Algorithm 2 Overall routing flow using RL policy

Require: A set of nets N and various design rules for a router
Ensure: Routing solution with optimized solution quality
1: Define M as the maximum number of iterations of RRR.
2: Define S as the set of nets' ordering scores.
3: $i \leftarrow 0$
4: **while** $i < M, N \neq \emptyset$ **do**
5: $i \leftarrow i + 1$
6: **for all** net $n \in N$ **do**
7: Extract net features f_n
8: **end for**
9: Use the RL policy π and features F to get the ordering scores of all nets S
10: batch list $B =$ **Scheduler**(N, S)
11: **for all** $b \in B$ **do**
12: Run maze routing, via selection and post-routing in multiple threads
13: **end for**
14: Calculate the total cost
15: **for all** $n \in N$ **do**
16: **if** n meet constraints **then**
17: Pop n from N
18: **else**
19: Rip-up n
20: **end if**
21: **end for**
22: **end while**

D. Overall Routing Flow

Once the policy is obtained, the overall routing flow is summarized in Algorithm 2. Once obtaining the ordering scores (line 9), we leverage Dr.CU to finish each RRR iteration (line 10-19). More specifically, we schedule all batches at the beginning of an RRR iteration (line 10) by sorting the nets according to the scores and divide them into batches, such that nets within a batch do not conflict with each other and can be routed in parallel to reduce the runtime [16]. If the RRR stopping criteria are not met, the iterations will continue until the maximum number of iterations is reached.

IV. Experimental Results

A. Experimental Setup

We define our environment using the OpenAI Gym interface with Dr.CU 2.0 [17] as the detailed router, and implement our RL agent network in PyTorch. All the experiments ran on a 64-bit Linux

TABLE II: Characteristics of ISPD 2018 & 2019 Contest Benchmarks

Benchmark	#std	#net	Die size (mm^2)	Tech. node (nm)
ISPD18_test1	8879	3153	0.20×0.19	45
ISPD18_test2	35 913	36 834	0.65×0.57	45
ISPD18_test3	35 973	36 700	0.99×0.70	45
ISPD18_test4	72 094	72 401	0.89×0.61	32
ISPD18_test5	71 954	72 394	0.93×0.92	32
ISPD18_test6	107 919	107 701	0.86×0.53	32
ISPD18_test7	179 865	179 863	1.36×1.33	32
ISPD18_test8	191 987	179 863	1.36×1.33	32
ISPD18_test9	192 911	178 857	0.91×0.78	32
ISPD18_test10	290 386	182 000	0.91×0.87	32
ISPD19_test1	8879	3153	0.148×0.146	32
ISPD19_test2	72 094	72 410	0.873×0.589	32
ISPD19_test3	8283	8953	0.195×0.195	32
ISPD19_test4	146 442	151 612	1.604×1.554	65
ISPD19_test5	28 920	29 416	0.906×0.906	65
ISPD19_test6	179 881	179 863	1.358×1.325	32
ISPD19_test7	359 746	358 720	1.581×1.517	32
ISPD19_test8	539 611	537 577	1.803×1.708	32
ISPD19_test9	899 341	895 253	2.006×2.151	32
ISPD19_test10	899 404	895 253	2.006×2.151	32

machine with two 20-core Intel Xeon@2.1 GHz CPUs and 64 GB RAM. We set the discount factor $\gamma = 0.99$, coefficient for the value loss $\beta = 0.25$, entropy cost $\eta = 0.001$, and learning rate to 0.001. We also set $\alpha = 0.1$ for the first 100 training episodes and reduce to 0 afterwards. A standard non-centered RMSProp is used as the gradient ascent optimizer. The neural network weights are initialized randomly. We use eight AC agents to train in parallel, and the maximum training time is set to 24 hours.

We experiment on the benchmarks from ISPD 2018 and ISPD 2019 Initial Detailed Routing Contests [4], [5]. The detailed information of the benchmarks are shown in Table II. We can see that benchmarks have quite different problem sizes, and technology nodes (32/45/65 nm). According to Dr.CU [17], the runtime of routing one of these benchmarks varies from two minutes to five hours. Ideally, it is expected to train and test a RL model on one technology node only. However, considering that most designs in Table II are in the 32 nm node, while the ones in 45/65 nm nodes are either too small or large, we choose a training dataset mixed with designs in 32 nm and 45 nm nodes, and test on the remaining to validate the framework. To balance the runtime overhead and universality of the generated model, `ISPD18_test3/5/6/7` are selected as benchmarks in the training dataset and the remaining sixteen as the test dataset. Due to `ISPD18_test7`'s big size, it is divided into many benchmarks with each region containing around 500 nets, and we choose two densest regions containing 7 and 26 violations to put in the training dataset. In conclusion, the training dataset contains {two regions clipped from `ISPD18_test7`, `ISPD18_test3/5/6`}. These training benchmarks have moderate and diverse sizes that can keep reasonable training time but also complicated enough to represent the real routing challenges.

B. Model performance

Table III and Table IV summarize the results of the training and testing datasets. We compare the wirelength, number of vias, DRC violations, total cost, and runtime between our RL framework and Dr.CU [17]. The violation values here are a summation of all the DRC violations mentioned in Equation (1). In the training dataset, with similar wirelength and number of vias, we can achieve 13% fewer DRC violations compared with the default policy in Dr.CU. The total cost only has small improvements. This is because the cost is dominated by wirelength due to its large scale according to its definition in the contests. The results on the training dataset indicate that our RL framework and training techniques are able to learn good policies from the benchmarks. We also observe around 6 % runtime overhead, which mostly comes from the feature extraction and the system integration between the `Python`-based RL agent and the `C++`-based Dr.CU implementation.

In the testing dataset, our policy can achieve an average of 14 % improvement in violations and 0.7 % in total cost without degradation in wirelength and number of vias. The results on the testing dataset demonstrate that the policy learnt from the training dataset can generalize to unseen benchmarks and achieve high-quality solutions on average.

One needs to mentioned that on large benchmarks like `ISPD19_test7-10` in 32 nm technology node, the RL policy can reduce the violations by 40 % to 50 %, which is rather promising. However, we observe that there are also outliers like `ISPD18_test4` and `ISPD19_test4` where the violations increase by 15 % and 46 %, respectively. The results of all the remaining benchmarks are either improved or within a comparable range. We speculate that the two outliers contain special features not in our training dataset or state space, causing unusual behaviors. `ISPD19_test4` is in 65 nm technology node with 6 metal layers, while the designs in the training dataset are in 45/32 nm technology nodes with 9 metal layers. These differences probably reduce the generalization performance of the RL policy in these two designs. In addition, `ISPD18_test4` has Metal2 obstacles in some of its standard cells, and `ISPD19_test6` has 16 block macros. These factors are likely to impact the routability of the designs, but not yet included in the state space. As our RL model is highly flexible, we can extract more features and conduct targeted training for specific benchmarks in a technology node to get better policies. This highlights the possibility of an online learning system to continuously improve the policy by running more and more benchmarks, which is worth exploring in the future. In this way, the policy network can evolve to be more robust and general on unseen benchmarks.

V. Conclusion

In this paper, we propose an asynchronous reinforcement learning framework to search for high-quality net ordering strategies in detailed routing automatically. We propose highly extensible agent models, mismatch penalty to enable efficient exploration of good policies. Experiments on ISPD 2018 & 2019 contest benchmarks demonstrate that our framework is able to learn an ordering policy that reduces the number of violations by 14 % on unseen benchmarks, compared with the state-of-the-art detailed router. The future work includes improving the agent network architecture to consider the correlation between multiple nets and expanding the state space to consider more features.

Acknowledgement

This work was supported by the National Natural Science Foundation of China (61874002, 62004006), Beijing Municipal Natural Science Foundation (2017ZX02315001), the National Key Research and Development Program of China (2019YFB2205005).

References

[1] H.-Y. Chen and Y.-W. Chang, "Global and detailed routing," in *Electronic Design Automation*. Elsevier, 2009, pp. 687–749.

TABLE III: Experimental Results on Training Dataset.

	Ours					Dr.CU 2.0 [17]				
Benchmarks	wirelengh	#via	violations	total cost	runtime (s)	wirelengh	#via	violations	total cost	runtime (s)
ISPD18_test3	8 936 901	318 445	342	5 276 358	189	8 937 031	318 470	361	5 285 974	172
ISPD18_test5	28 702 560	965 402	388	16 476 084	638	28 702 043	965 503	393	16 478 528	610
ISPD18_test6	37 006 881	1 480 649	63	21 496 288	793	37 004 026	1 480 634	95	21 511 021	756
ISPD18_test7	67 276 888	2 402 960	735	38 811 801	1576	67 273 394	2 402 970	792	38 838 412	1466
Geomean Ratio	1.000	1.000	0.870	0.999	1.067	1.000	1.000	1.000	1.000	1.000

TABLE IV: Experimental Results on Testing Dataset.

	Ours					Dr.CU 2.0 [17]				
Benchmarks	wirelengh	#via	violations	total cost	runtime (s)	wirelengh	#via	violations	total cost	runtime (s)
ISPD18_test1	453 511	32 363	0.3	291 631	10	453 432	32 360	0.6	291 736	9
ISPD18_test2	8 059 053	325 503	4	4 682 549	136	8 058 836	325 554	1	4681051	125
ISPD18_test4	26 979 639	729 092	584	15 240 004	745	26 978 758	728 433	507	15 199 745	694
ISPD18_test8	67 593 559	2 413 037	756	39 000 616	1541	67 588 741	2 412 377	819	39 028 574	1474
ISPD18_test9	56 899 506	2 411 528	54	33 299 758	1260	56 894 634	2 410 568	139	33 337 715	1196
ISPD18_test10	70 396 465	2 593 585	9165	44 967 988	2373	70 397 656	2 594 731	8271	44 523 940	2244
ISPD19_test1	669 593	36 053	1110	961 903	109	669 855	36 093	1121	967 613	102
ISPD19_test2	25 647 830	788 668	4333	16 567 801	1610	25 631 985	787 174	4262	16 521 566	1499
ISPD19_test3	899 746	65 062	96	627 997	54	898 890	64 895	167	662 485	52
ISPD19_test4	31 266 702	1 030 982	7968	21 679 190	1653	31 252 265	1 031 270	5455	20 415 922	1548
ISPD19_test5	4 878 661	153 501	426	2 959 333	163	4 878 631	153 590	408	2 950 245	154
ISPD19_test6	67 776 361	1 987 985	9474	42 601 150	3340	67 760 199	1 986 157	8944	42 324 412	3158
ISPD19_test7	127 176 480	4 809 608	7798	77 106 306	8338	127 185 209	4 811 539	11 649	79 040 133	7812
ISPD19_test8	195 596 773	7 331 049	9128	117 024 322	11 870	195 588 815	7 330 041	16 291	12 060 0077	11 089
ISPD19_test9	297 319 642	12 195 824	16 745	181 424 169	15 967	297 300 480	1 218 9423	34 632	19 034 4949	15 225
ISPD19_test10	294 552 415	12 490 155	18 150	181 331 330	17 108	294 515 790	1 248 2849	32 743	18 859 4992	16 156
Geomean Ratio	1.000	1.000	0.856	0.993	1.065	1.000	1.000	1.000	1.000	1.000

[2] J. Xu, X. Hong, T. Jing, Y. Cai, and J. Gu, "An efficient hierarchical timing-driven steiner tree algorithm for global routing," *Integration*, vol. 35, no. 2, pp. 69–84, 2003.

[3] W.-H. Liu, W.-C. Kao, Y.-L. Li, and K.-Y. Chao, "Nctu-gr 2.0: Multithreaded collision-aware global routing with bounded-length maze routing," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 32, no. 5, pp. 709–722, 2013.

[4] S. Mantik, G. Posser, W.-K. Chow, Y. Ding, and W.-H. Liu, "Ispd 2018 initial detailed routing contest and benchmarks," in *Proceedings of the 2018 International Symposium on Physical Design*, 2018, pp. 140–143.

[5] W.-H. Liu, S. Mantik, W.-K. Chow, Y. Ding, A. Farshidi, and G. Posser, "Ispd 2019 initial detailed routing contest and benchmark with advanced routing rules," in *Proceedings of the 2019 International Symposium on Physical Design*, 2019, pp. 147–151.

[6] M. M. Ozdal and M. D. Wong, "Archer: A history-based global routing algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 4, pp. 528–540, 2009.

[7] T.-H. Wu, A. Davoodi, and J. T. Linderoth, "Grip: Global routing via integer programming," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 1, pp. 72–84, 2010.

[8] M. M. Ozdal, "Detailed-routing algorithms for dense pin clusters in integrated circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 3, pp. 340–349, 2009.

[9] T. Nieberg, "Gridless pin access in detailed routing," in *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2011, pp. 170–175.

[10] X. Xu, Y. Lin, V. Livramento, and D. Z. Pan, "Concurrent pin access optimization for unidirectional routing," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2017, pp. 1–6.

[11] M. Ahrens, M. Gester, N. Klewinghaus, D. Müller, S. Peyer, C. Schulte, and G. Tellez, "Detailed routing algorithms for advanced technology nodes," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 34, no. 4, pp. 563–576, 2014.

[12] Q. Ma, H. Zhang, and M. D. Wong, "Triple patterning aware routing and its comparison with double patterning aware routing in 14nm technology," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 591–596.

[13] Y.-H. Lin, B. Yu, D. Z. Pan, and Y.-L. Li, "Triad: A triple patterning lithography aware detailed router," in *Proceedings of the International Conference on Computer-Aided Design*, 2012, pp. 123–129.

[14] Z. Liu, C. Liu, and E. F. Young, "An effective triple patterning aware grid-based detailed routing approach," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015, pp. 1641–1646.

[15] A. B. Kahng, L. Wang, and B. Xu, "TritonRoute: an initial detailed router for advanced VLSI technologies," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[16] G. Chen, C.-W. Pui, H. Li, and E. F. Young, "Dr. CU: Detailed Routing by Sparse Grid Graph and Minimum-Area-Captured Path Search," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[17] H. Li, G. Chen, B. Jiang, J. Chen, and E. F. Y. Young, "Dr. CU 2.0: A Scalable Detailed Routing Framework with Correct-by-Construction Design Rule Satisfaction," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Westminster, CO, USA: IEEE, Nov. 2019, pp. 1–7.

[18] S. M. Gonçalves, L. S. Rosa, and F. S. Marques, "Draps: A design rule aware path search algorithm for detailed routing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2019.

[19] G. Liu, Z. Zhuang, W. Guo, and T.-C. Wang, "Rdta: An efficient routability-driven track assignment algorithm," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 2019, pp. 315–318.

[20] F.-K. Sun, H. Chen, C.-Y. Chen, C.-H. Hsu, and Y.-W. Chang, "A multithreaded initial detailed routing algorithm considering global routing guides," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–7.

[21] H. Liao, Q. Dong, X. Dong, W. Zhang, W. Zhang, W. Qi, E. Fallon, and L. B. Kara, "Attention routing: track-assignment detailed routing using attention-based reinforcement learning," *arXiv preprint arXiv:2004.09473*, 2020.

[22] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," *arXiv:1602.01783 [cs]*, Jun. 2016, arXiv: 1602.01783.