Asynchronous Reinforcement Learning Framework and Knowledge Transfer for Net Order Exploration in Detailed Routing

Yibo Lin Member, IEEE, Tong Qu, Zongqing Lu, Yajuan Su, Yayi Wei

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 automatically search for optimal ordering strategies and a transfer learning (TL) algorithm to improve performance. By asynchronous querying, the router, pre-training the RL agents, and finetuning with the TL algorithm, we can generate high-performance routing sequences to achieve a 26% reduction in the DRC violations and a 1.2% reduction in the total costs compared with the state-of-the-art detailed router.

Index Terms—Physical design, detailed routing, reinforcement learning, transfer learning, policy distillation

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 [3], [6], [7]. 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]–[10]. Ahrens et al. explored specific data structures for efficient detailed routing [11]. Manufacturing constraints have also been explored in [12]–[14], such as 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]–[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], [21] 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 [22].

Among the aforementioned detailed routers, most of them substantially follow the sequential routing strategy with the negotiation-based rip-up and reroute scheme [16]–[18], [20]. 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. For instance, some net ordering indicators are listed here: 1) the number of pins in a net; 2) the number of DRC violations caused by a net [23]; 3) the region size covered by a net [17]; 4) the distance from a certain point [24]. In addition, the net order may change according to the current routing status and historical penalties during the rip-up and reroute stage [3]. The performance of these ordering strategies 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 contributions can be summarized as follows.

- We develop an asynchronous reinforcement learning framework to learn the ordering strategy in sequential detailed routing. By developing a customized neural network architecture, we can apply the learned model to different designs.
- We propose a transfer learning algorithm that can adapt the pre-trained policy to target designs by finetuning small, clipped regions for better performance.
- Experimental results on ISPD 2018 & 2019 contest benchmarks [4], [5] demonstrate that the ordering strategy obtained from our framework generalizes well. Compared with the state-of-the-art detailed router Dr.CU 2.0 [17], the DRC violations and the total costs are reduced by 14% and 0.7%, respectively. With transfer learning, we can reduce the DRC violations and the total costs by 26% and 1.2%, respectively.

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

II. PRELIMINARIES

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

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. 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

![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.](image)

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, which will be explained in detail in Section III-A and Table I.

C. Reinforcement Learning

Machine Learning (ML) is an effective practical tool to optimize a design in the absence of suitable models for optimization. One of the main obstacles in using supervised ML-based techniques for solving routing problems, especially the net ordering problems, is the lack of golden labeled datasets to learn. One way to overcome this problem is to use a reinforcement learning approach. RL has been successfully applied in many applications. A typical RL problem can be considered as training an agent to interact with an environment. As illustrated
in Fig. 4, at each step $t$, the agent $g$ observes a state $s_t \in S$, takes an action $a_t \in A$ based on $s_t$, receives a reward $r_t \in \mathbb{R}$, and then the state stochastically transits to the next state $s_{t+1}$. The objective is to learn a policy $\pi(a_t|s_t)$ that maximizes the expected cumulative reward $R = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$, starting from any state $s$, where scalar $\gamma \in (0, 1]$ is a discount rate.

In this work, we define the environment as the router, and the design to be routed, 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.

### 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 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.

We further define the transfer learning problem to adapt the net ordering policy to a target design.

**Problem 2 (Transfer learning).** Given a target design with a set of nets $N$ and a pre-trained policy for net ordering, finetune the net ordering policy with small clipped regions of the target design. The performance metrics, as mentioned earlier, after routing the target design can be optimized.

### III. REINFORCEMENT LEARNING FRAMEWORK

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.

#### A. Basic RL Setup

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

**State space $S$:** 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 $A$:** An action $a$ is a real number vector. Each number is defined as an ordering score of a net.

**Reward $R$:** 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:

$$ r = -c + c_o \tag{1} $$

Where $c$ and $c_o$ 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:

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

### TABLE I: Features of Each Net.

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

**Feature Dimension Description**

- 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.

...
Intuitively, the policy network tells us the ordering scores of action by sampling from this normal distribution \( \pi(a_t | s_t; \theta_v) \) as the probability of the sampled action \( a_t \). We pick the action from this distribution \( p \sim N(\mu, \sigma^2) \) over the actions. 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 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 \( E \times 64 \) input feature tensor into an \( E \times 64 \) tensor. The policy network takes this tensor and \( 64 \times 64 \) hidden units to obtain a scalar at the output. The value network outputs the value function \( V(s_t; \theta_v) \) for the state \( s_t \) for non-terminal \( s_t \) and \( t \). 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.

D. 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:

\[ r = -c + c_0 - \frac{\alpha}{k} \sum_{i=1}^{k} \Delta a_i^2 \]  

(3)

Where \( \Delta a \) is the difference between the predicted ordering scores and the sizes of routing regions, \( \alpha \) is a user-defined parameter, and \( k \) is the number of nets to be routed. The parameter \( \alpha \) is positive only at the early training steps and then set to zero. The detailed setup can be found in Section V.

Fig. 7 compares the learning speeds of the two reward function definitions. 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. In other words, the agent is free to generate different orders from that in Dr.CU.

IV. TRANSFER LEARNING ALGORITHM

In this section, we first introduce the transfer learning algorithm based on policy distillation. In the end, we summarize the overall flow of our TL algorithm.

We assume a reasonably good policy is available. Our task is to mine the knowledge from the pre-trained policy and adapt to a target design to improve the performance.

A. Policy Distillation Algorithm

In the previous section, we aim at training a general RL policy to solve the net ordering tasks of multiple designs, which is essentially a multi-task learning problem. However, it is not always easy to achieve good generality across a wide range of designs. On designs with unique characteristics, a generally good policy may not capture the unique design styles and eventually fails to result in high solution quality. If we can customize the policy for each design with low overhead, there is an opportunity to improve the performance further. To reduce the overhead of customization, we finetune the well-trained policy from the previous section using a small region clipped from the target design instead of training from scratch. In this way, we can minimize the transfer learning overhead by repeatedly routing the target design when the agents interact with the environment. The key for transfer learning is to effectively transfer the knowledge from the source domain (i.e., the pre-trained policy) to the target domain (i.e., for routing the target design).

Policy distillation is a transfer learning approach that distills the knowledge from a teacher network to a student network. Typical RL policy distillation frameworks transfer the teacher policy in a supervised learning paradigm. Specifically, a student policy is learned by minimizing the Kullback-Leibler (KL) divergence of actions between the teacher policy \( \pi^T \) and student policy \( \pi^S \) [26]. As explained in Section III-C, one action \( a \) is sampled from the normal distribution \( p \) from the policy network, so the policy distillation can be completed by optimizing the per-time-step KL divergence between the distributions of the teacher and the student over actions:

\[ L_{KL}(D, \theta^S) = \sum_{t=1}^{D} p^T_t \log \frac{p^T_t}{p^S_t}, \]  

(4)

where \( p^T_t \) is the normal distribution from the teacher network at step \( t \), \( p^S_t \) is the distribution from the student network of this step, \( \theta^S \) denotes the parameters of the student policy network, and \( D \) is a set of distributions from the teacher network.

B. Policy Transfer Flow

As our network structure in Section III-C is decoupled with the number of nets, different designs’ action spaces can
be considered as the same dimensions. Hence, the student network can directly mimic the action pattern of the teacher network.

Based on the above observation, we propose a policy distillation flow to transfer knowledge between designs. As illustrated in Fig. 8, the distribution $p$ of the teacher network from a source design (environment) $E^T$ is used as our supervision to update the student network of target design (environment) $E^S$. This procedure is performed by minimizing Equation (4) over student policy network parameters $\theta^S$ for $E^S$, $\theta^S$ including the net feature encoder, value layer, and policy layer. The input to both the teacher and the student networks is the state $s$ of environment $E^S$. It means that we want to transfer the expertise from the environment $E^T$ towards the current state. Symbol $D$ is a set of distributions from the teacher network in one batch.

Suppose we have already had a reasonably good policy for most designs. We want to use the knowledge from the policy to improve the performance of a particular design further. In the proposed transfer learning approach, we first select a clip from the target design because training on a complete design is not affordable, mainly when the design contains tens of thousands of nets. Given a representative clip, we observe that training can significantly reduce the training time without much performance loss. Then, we train the student network by interacting with the clip (environment) $E^S$ for a certain amount of steps, e.g., 1000 steps in the experiments. Afterward, we start performing policy distillation.

C. Overall Flow

Fig. 9 shows the overall flow. The A3C algorithm is first used to train on multiple designs to obtain the pre-trained model $M^T$. For each input design $d_i$ to be routed, we select a clip $l_i$ from it. The policy distillation is performed to guide the model $M_i^S$, which is trained on the clip $l_i$. After a certain period of training, the best model for clip $l_i$ is selected. Finally, the input design $d_i$ is routed under the guidance of model $M_i^S$.

The overall routing flow is summarized in Algorithm 2, given that the RL model $M^S$ to determine the net ordering. We first extract features for each net to be routed within each routing iteration and then obtain the ordering scores using the RL policy (line 9). After that, 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 dividing them into batches. Nets within a batch that do not conflict with each other can be routed 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. We keep the same settings as Dr.CU except for the net ordering.

V. 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 machine with two 20-core Intel Xeon@2.1 GHz CPUs and 64 GB RAM.

The latest design benchmarks available to academics are the ISPD 2018 and ISPD 2019 Initial Detailed Routing Contests [4], [5] benchmarks, which have 20 designs. We experiment on those benchmarks. The detailed information of the benchmarks
Algorithm 3 Clip Selection
1: Define \( H_C \) and \( W_C \) as the initial width and height of a clip \( C \), and \( N_C \) as the step size.
2: while the number of nets in clip \( C \) is smaller than 500 do
3: Divide the benchmarks into clips with sizes of \( H_C \times W_C \)
4: Choose the clip with the highest pin density as the candidate clip \( C \)
5: Project pins outside the clip \( C \) to the boundary
6: \( W_C \leftarrow W_C + N_C \)
7: \( H_C \leftarrow H_C + N_C \)
8: end while
9: Add the clip \( C \) into the training dataset

is shown in Table II. We can see that benchmarks have quite different problem sizes, and technology nodes (32/45/65 nm).

As the benchmarks are large, we select clips with suitable scales and add them to the training dataset. The procedure of clip selection is described in Algorithm 3. The idea is to select a dense clip including around 500 nets because we observe this is an affordable scale to invoke the router during training repeatedly. We first divide a benchmark into fixed-size clips (line 3) and then choose the densest one in terms of pins (line 4). We then project the pins outside the clip to its boundary (line 5). If the number of nets in the clip exceeds 500, we add it to the training dataset. Otherwise, we expand the clip to include more nets and repeat the above steps.

B. Asynchronous Advantage Actor-Critic

In this subsection, we verify the effectiveness of the A3C framework. We set the discount factor \( \gamma = 0.99 \), the 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 afterward. 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 (around 8000 episodes).

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 large size, we choose two clips from it 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.

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 mention 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. At this point, we have obtained a model that can perform well in most design, and further improvement will be completed in transfer learning.

C. Transfer Learning via Policy Distillation

In this subsection, we design experiments to illustrate the effectiveness of the policy distillation algorithm. We obtained a generally good model in previous experiments, but there are some outliers like ISPD18_test4 and ISPD19_test4. This result indicates that the knowledge gained in the training dataset has insufficient generalization capabilities. Therefore, we transfer the knowledge from the pre-trained policy to the student network for each design via policy distillation. One challenge is that the routing of each design is time-consuming in the testing dataset. As we mentioned before, for each design \( d_i \) to be tested, we select a clip \( l_i \) from it for interaction. The clip \( l_i \) must have sufficient representation. We select a clip containing about 500 nets from the region with the highest pin density such that the routing can be finished within 1 minute by Dr.CU. Pin density is a straightforward metric for clip selection. We leave the exploration of more complicated clipping strategies to the future. In the following experiments, we maintain consistent hyper-parameters of the network structures across all designs.

The student agents are trained with the clip \( l_i \) only and perform policy distillation after 1000 episodes. We set the maximum time for the entire training to three hours (assuming the maximum time available for transfer learning, equivalently around 3000 episodes). Finally, the model is evaluated with the design \( d_i \). Table IV shows the comparison of wirelength, number of vias, number of DRC violations, and total costs between Dr.CU, the RL algorithm [27], and the transfer learning algorithm. By enabling transfer learning, we can further reduce the average number of DRC violations by 12% and the average total cost by 0.5% compared with the RL algorithm, while other metrics remain almost the same as the RL algorithm. Note that these numbers are an extra improvement from the RL algorithm. Compared with Dr.CU, the TL algorithm can contribute to 26% reduction of DRC violations and 1.2% reduction of total cost.

To further verify the performance of the model with additional training iterations, we also train the model for 100K episodes on ISPD19_test2 and ISPD19_test8, and show the training curves in Fig. 10. As routing an entire design to obtain the number of DRC violations is time-consuming, we only sample every 1000 and 4000 episodes for these designs, respectively. We can see that the performance of the TL curves continues to improve and gradually saturates at 60K to 100K episodes. We also plot the curves of training from scratch using the RL algorithm as ‘RL-S’ in the figure. Compared with the RL-S curve, the TL curve drops much faster, indicating the effectiveness of the proposed transfer learning technique compared with training from scratch. We also observe that the #violation of the average TL and RL-S curves can go much higher than that of the RL curve between 0 to 20K episodes. This is because the performance oscillates greatly at the beginning of the training iterations, several extremely poor points cause large average number of violations. With the training continuing, the performance gradually converges and becomes more stable.

In the previous section, we observe that the numbers of DRC violations for ISPD18_test2/4/10 and ISPD19_test2/4/5/6 are larger than Dr.CU’s results. With our transfer learning technique, ISPD18_test4 and ISPD19_test2 can outperform Dr.CU, while the results for ISPD18_test2/10 and ISPD19_test4/5/6 are still not as good as Dr.CU, especially that the results of ISPD18_test10 and ISPD19_test4/5/6 are far from expected. One possible reason lies in insufficient training, and the teacher network guides the student networks in the wrong direction. To verify that, we continue the training iterations of transfer learning on these designs and report the numbers of DRC violations with the intermediate policies, as shown in Fig. 11. Each model is trained for 100K to 140K episodes (around 100 hours to 140 hours). Similar to that in Fig. 10, we only sample every 500, 1000, and 2000 episodes for each of the four designs mentioned above according to their sizes, respectively. We compare the transfer learning curves with the results of Dr.CU and the RL algorithm as well as the results of training from scratch (‘RL-S’). Fig. 11 shows that in the first 3000 episodes (about three hours), the model’s performance improves slowly. When the training continues, the performance of the models keeps improving, even though
### TABLE III: Experimental Results on the Training Dataset (Comparison between Dr.CU [17] and RL [27]).

<table>
<thead>
<tr>
<th>Design</th>
<th>Wirelength (10^7)</th>
<th>#via (10^6)</th>
<th>#vio</th>
<th>Cost (10^7)</th>
<th>T (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dr.CU RL</td>
<td>Dr.CU RL</td>
<td>Dr.CU RL</td>
<td>Dr.CU RL</td>
<td>Dr.CU RL</td>
</tr>
<tr>
<td>test3</td>
<td>0.894 0.894</td>
<td>0.318 0.318</td>
<td>361 342</td>
<td>0.529 0.528</td>
<td>172 189</td>
</tr>
<tr>
<td>test5</td>
<td>2.870 2.870</td>
<td>0.966 0.965</td>
<td>393 388</td>
<td>1.648 1.648</td>
<td>610 638</td>
</tr>
<tr>
<td>test6</td>
<td>3.700 3.701</td>
<td>1.481 1.481</td>
<td>95 63</td>
<td>2.151 2.150</td>
<td>756 793</td>
</tr>
<tr>
<td>test7</td>
<td>6.727 6.728</td>
<td>2.403 2.403</td>
<td>792 735</td>
<td>3.884 3.881</td>
<td>1466 1576</td>
</tr>
</tbody>
</table>

### TABLE IV: Experimental Results on the Testing Dataset (Comparison between Dr.CU [17], RL [27], and TL).

<table>
<thead>
<tr>
<th>Design</th>
<th>Wirelength (10^7)</th>
<th>#via (10^6)</th>
<th>#vio</th>
<th>Cost (10^7)</th>
<th>T (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dr.CU RL TL</td>
<td>Dr.CU RL TL</td>
<td>Dr.CU RL TL</td>
<td>Dr.CU RL TL</td>
<td></td>
</tr>
<tr>
<td>test1</td>
<td>0.05 0.05 0.05</td>
<td>0.03 0.03 0.03</td>
<td>1 0 0</td>
<td>0.03 0.03 0.03</td>
<td>9 10 10</td>
</tr>
<tr>
<td>test2</td>
<td>0.81 0.81 0.81</td>
<td>0.33 0.33 0.33</td>
<td>1 4 4</td>
<td>0.47 0.47 0.47</td>
<td>125 136 137</td>
</tr>
<tr>
<td>test3</td>
<td>2.70 2.70 2.70</td>
<td>0.73 0.73 0.73</td>
<td>507 584 487</td>
<td>1.52 1.52 1.52</td>
<td>696 745 727</td>
</tr>
<tr>
<td>test4</td>
<td>6.76 6.76 6.76</td>
<td>2.41 2.41 2.41</td>
<td>819 756 753</td>
<td>3.90 3.90 3.90</td>
<td>1474 1541 1548</td>
</tr>
<tr>
<td>test5</td>
<td>5.69 5.69 5.69</td>
<td>2.41 2.41 2.41</td>
<td>139 54 9</td>
<td>3.33 3.33 3.33</td>
<td>1196 1260 1240</td>
</tr>
<tr>
<td>test10</td>
<td>7.04 7.04 7.04</td>
<td>2.59 2.59 2.60</td>
<td>8271 9165 8915</td>
<td>4.45 4.50 4.49</td>
<td>2244 2373 2315</td>
</tr>
</tbody>
</table>

### Relative Ratios

<table>
<thead>
<tr>
<th>Design</th>
<th>Wirelength</th>
<th>#via</th>
<th>#vio</th>
<th>Cost</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dr.CU RL</td>
<td>Dr.CU RL</td>
<td>Dr.CU RL</td>
<td>Dr.CU RL</td>
<td>Dr.CU RL</td>
</tr>
<tr>
<td>test3</td>
<td>1.000 1.000</td>
<td>1.000 1.000</td>
<td>1.000 0.947</td>
<td>1.000 0.998</td>
<td>1.000 1.099</td>
</tr>
<tr>
<td>test5</td>
<td>1.000 1.000</td>
<td>1.000 0.999</td>
<td>1.000 0.987</td>
<td>1.000 1.000</td>
<td>1.000 1.046</td>
</tr>
<tr>
<td>test6</td>
<td>1.000 1.000</td>
<td>1.000 1.000</td>
<td>1.000 0.663</td>
<td>1.000 1.000</td>
<td>1.000 1.049</td>
</tr>
<tr>
<td>test7</td>
<td>1.000 1.000</td>
<td>1.000 1.000</td>
<td>1.000 0.928</td>
<td>1.000 0.999</td>
<td>1.000 1.075</td>
</tr>
<tr>
<td>Min.</td>
<td>1.000 1.000</td>
<td>1.000 0.999</td>
<td>1.000 0.663</td>
<td>1.000 0.998</td>
<td>1.000 1.046</td>
</tr>
<tr>
<td>Max.</td>
<td>1.000 1.000</td>
<td>1.000 1.000</td>
<td>1.000 0.987</td>
<td>1.000 1.000</td>
<td>1.000 1.099</td>
</tr>
<tr>
<td>Geo. Mean</td>
<td>1.000 1.000</td>
<td>1.000 1.000</td>
<td>1.000 0.870</td>
<td>1.000 0.999</td>
<td>1.000 1.067</td>
</tr>
</tbody>
</table>

### Relative Ratios

<table>
<thead>
<tr>
<th>Design</th>
<th>Wirelength</th>
<th>#via</th>
<th>#vio</th>
<th>Cost</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dr.CU RL TL</td>
<td>Dr.CU RL TL</td>
<td>Dr.CU RL TL</td>
<td>Dr.CU RL TL</td>
<td></td>
</tr>
<tr>
<td>test1</td>
<td>0.07 0.07 0.07</td>
<td>0.04 0.04 0.04</td>
<td>1121 1110 1045</td>
<td>0.10 0.10 0.09</td>
<td>102 109 106</td>
</tr>
<tr>
<td>test2</td>
<td>2.56 2.56 2.56</td>
<td>0.79 0.79 0.79</td>
<td>4262 4333 4253</td>
<td>1.65 1.66 1.65</td>
<td>1499 1610 1625</td>
</tr>
<tr>
<td>test3</td>
<td>0.09 0.09 0.09</td>
<td>0.06 0.06 0.07</td>
<td>167 96 77</td>
<td>0.07 0.06 0.06</td>
<td>52 54 54</td>
</tr>
<tr>
<td>test4</td>
<td>3.13 3.13 3.13</td>
<td>1.03 1.03 1.03</td>
<td>5455 7968 7949</td>
<td>2.04 2.17 2.17</td>
<td>1548 1653 1665</td>
</tr>
<tr>
<td>test5</td>
<td>0.49 0.49 0.49</td>
<td>0.15 0.15 0.15</td>
<td>408 426 436</td>
<td>0.30 0.30 0.30</td>
<td>154 163 165</td>
</tr>
<tr>
<td>test6</td>
<td>6.78 6.78 6.78</td>
<td>1.99 1.99 1.98</td>
<td>8944 9474 9363</td>
<td>4.23 4.26 4.25</td>
<td>3518 3340 3374</td>
</tr>
<tr>
<td>test7</td>
<td>12.72 12.72 12.71</td>
<td>4.81 4.81 4.80</td>
<td>11649 7798 7577</td>
<td>7.90 7.71 7.69</td>
<td>7812 8338 8379</td>
</tr>
<tr>
<td>test8</td>
<td>19.56 19.56 19.55</td>
<td>7.33 7.33 7.32</td>
<td>16291 9128 8676</td>
<td>12.06 11.70 11.67</td>
<td>11089 11870 11855</td>
</tr>
<tr>
<td>test9</td>
<td>29.73 29.73 29.72</td>
<td>12.19 12.20 12.17</td>
<td>34632 16745 16015</td>
<td>19.03 18.14 18.10</td>
<td>15225 15967 15681</td>
</tr>
<tr>
<td>test10</td>
<td>29.46 29.45 29.45</td>
<td>12.48 12.49 12.49</td>
<td>32743 18150 17522</td>
<td>18.86 18.13 18.10</td>
<td>16156 17108 17335</td>
</tr>
<tr>
<td>Min.</td>
<td>1.000 1.000 1.000</td>
<td>1.000 1.000 1.000</td>
<td>1.000 1.000 1.000</td>
<td>1.000 1.000 1.000</td>
<td></td>
</tr>
<tr>
<td>Max.</td>
<td>1.000 1.000 1.000</td>
<td>1.000 1.000 1.000</td>
<td>1.000 1.000 1.000</td>
<td>1.000 1.000 1.000</td>
<td></td>
</tr>
<tr>
<td>Geo. Mean</td>
<td>1.000 1.000 1.000</td>
<td>1.000 1.000 1.000</td>
<td>1.000 1.000 1.000</td>
<td>1.000 1.000 1.000</td>
<td></td>
</tr>
</tbody>
</table>
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2021.3117505, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 10

VI. 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 and 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. We also propose a transfer learning algorithm to further improve the agent models' performance based on policy distillation. The models after transfer learning can reduce the number of violations by 26% on unseen benchmarks, compared with the state-of-the-art detailed router. We also observe that the transfer learning technique eventually leads to better convergence than training from scratch on these designs.

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. We also plan to explore techniques [28] to handle the staleness between global and local agents in the asynchronous reinforcement learning framework.

REFERENCES


