
NVCell: Generate Standard Cell Layout in Advanced Technology Nodes with Reinforcement Learning

Haoxing Ren
NVIDIA Corporation
Austin, TX
haoxingr@nvidia.com

Matthew Fojtik
NVIDIA Corporation
Durham, NC
mfojtik@nvidia.com

Brucek Khailany
NVIDIA Corporation
Austin, TX
bkhailany@nvidia.com

Abstract

Standard cell layout in advanced technology nodes are done manually in the industry today. Automating standard cell layouts is challenging because of the exploding number and complexity of design rule checking (DRC), especially when the design goal is to minimize cell area. In this paper we propose a machine learning based approach to handle DRC constraints. In our approach, we apply a genetic algorithm to create routing candidates and use reinforcement learning (RL) to fix the design rule violations incrementally. A design rule checker provides feedback on violations to the RL agent and the agent learns how to fix them based on the data. This approach is also applicable to future technologies with unseen DRCs. Based on this approach, we built a layout generator called **NVCell** that includes a device placer based on a simulated annealing method and a router based on a genetic algorithm and reinforcement learning. **NVCell** can generate layouts with equal or lower area for over 75% of cells in an industry standard cell library.

1 Introduction

Standard cells are the building blocks of digital VLSI design. Modern designs are constructed from hundreds of millions of instances of standard cells. Large semiconductor companies and intellectual property providers often have dedicated teams designing standard cell libraries for each technology node. Each library usually consists of thousands of standard cells. The cell design objective is minimizing cell width (cell height is fixed for each library) to improve area efficiency.

Today most industrial standard cell designs are still done manually. Automating standard cell layout can not only speed up the design process, but also enable design and technology co-optimization (DTCO), which simultaneously optimizes standard cells and chip designs to achieve better performance.

Standard cell layout design automation includes two major steps: placement and routing. The placement step places devices and assigns pin locations; the routing step connects device terminals and pins based on net connectivity. The routing step is the more important and difficult step because routing needs to satisfy design rule checking (DRC) constraints. In advanced technology nodes, not only are the number of design rules exploding, but rules also become more complex. Most of the new complexity comes from rules that involve multiple layout shapes that were previously independent of each other. Although mathematical optimization approaches based on Boolean satisfiability problem (SAT) [14] and mixed-integer linear program (MILP) [17] were proposed to automate routing and achieved good results, these techniques depend on the assumption that all design rule constraints can be expressed in the forms such as conjunctive normal form for SAT or linear inequality for MILP. It is not clear whether all the DRCs can be expressed efficiently in these forms. A large number of constraints will have to be created to handle all DRCs, which makes it difficult to scale to larger designs. Furthermore, these constraints will have to be rewritten by hand for every new technology node or standard cell layout template.

Preprint. Under review.

Reinforcement learning (RL) has achieved impressive super-human performance in solving many *game* like problems. We are interested in whether RL can also achieve super-human performance in the routing task. We believe that the main advantage of RL over other mathematical optimization methods such as SAT and MILP is that it does not require analytical formulation of the DRCs. The constraints can be enforced by the reward given by a environment where DRC analysis runs independently from the optimization. [9] proposed to apply Reinforcement Learning to the routing problem directly, i.e. make the RL agent create routing actions for each wire where the action space is a routing action (North, South, West, East) for each net. Instead of making the RL agent learn the job of a maze router, which can be solved by many efficient algorithms [16], we propose to make it learn how to fix DRCs on existing routes. We decompose the routing problem into two independent steps: routing and DRC fixing. The sub-problem of DRC fixing is relatively easier for RL to learn and it scales to large designs since DRC problems are local, while general routing requires global information, especially for long routes.

Our key contributions are listed below:

- We build an automated layout generator: **NVCell** generates competitive layouts for over 75% of cells in an industrial cell library on an advanced technology node. The widths of generated cells are equal or less than the manual design.
- We propose a Reinforcement Learning based method to fix DRC errors given existing routes on standard cells. Trained on one standard cell, the model is transferable to all the standard cells we have tested in our library. The model can be further retrained on each cell to improve the results.
- We propose a simulated annealing based algorithm for device placement and pin assignment. It performs both device pairing and placement concurrently to find an optimum placement.
- We propose a genetic algorithm based routing flow to find minimum routes and optimize the DRC errors. Together with the RL based DRC fixer, it found routable DRC-clean cell layouts with reduced widths compared to the best layouts found by expert layout engineers.

2 Prior Art

Prior placement techniques include heuristic based methods, exhaustive search based methods, and mathematical programming based methods. The heuristic based methods [19] first find all possible chains in the circuit, i.e. devices that can share diffusions consecutively, and then select a number of chains that cover all the devices. The exhaustive search based methods [17] [8] go through all possible device placement configurations and might use branch and bound or dynamic programming techniques to speedup the search process. The mathematical programming based methods [11] [7] leverage MILP or SAT algorithms to find optimal device placement. It has been shown that these placement techniques can generate good placement solutions. However, for our implementation, we prefer a technique that can adapt to custom layout constraints and is also easy to implement. Therefore we propose to use simulated annealing to perform placement.

Prior routing techniques include channel routing, SAT, and MILP based routing methods. Channel routing algorithms have been proposed to do standard cell routing early on [19]. However, commonly used deterministic channel routing methods such as LEA, Dogleg, Greedy, YACR2, etc. [16], only generate a particular routing solution and do not handle DRCs well. SAT based routing [14] creates candidate routes for each terminal pair and leverages SAT to find feasible routing candidates for all terminal pairs. It requires DRC checks to prune all conflicting routing candidates. The quality of routing candidates also limits the final routing quality. Therefore it often can not find routing solutions for complicated cells. MILP based routing methods [17][8] formulate the routing problem as a mixed integer linear programming problem. It is capable of routing very complex cells in a 7nm process technology. This method, however, relies on a MILP solver to solve a large number of constraints and requires DRCs to be expressed in conditional equality or inequality form. This makes supporting newer technology nodes difficult. [6] uses a combination of MILP and rip-up-reroute techniques to route, which would have similar issues to those mentioned previously.

RL has achieved great successes in playing games such as Atari and GO. Recently RL has also been proposed to handle placement and routing problems. [12] leverages RL to place macros. [10] uses Deep Q Learning (DQN) [13] to create routing direction action, i.e. going north, south, etc at each step. [9] uses the attention model-based REINFORCE[18] algorithm to select routing orders and uses a pattern router to generate actual routes once a routing order is determined. [3] leverages both Monte carlo tree search (MCTS) and neural network based directional action to find routes. Most of

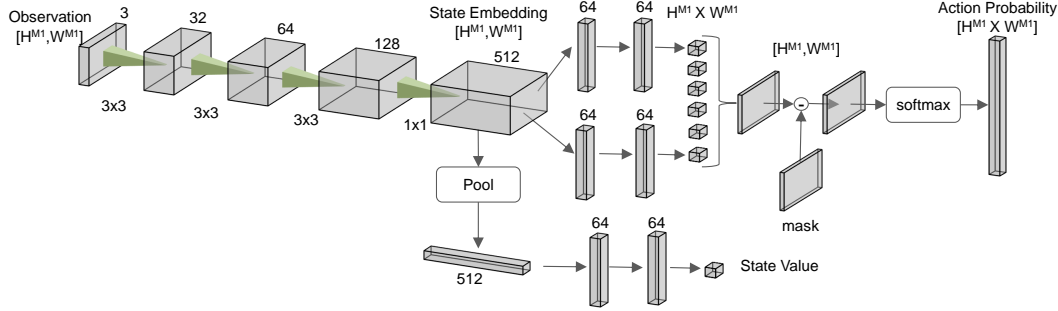


Figure 1: DRC RL Model Architecture

these approaches only aim to connect the routes without consideration of design rule violations. It is not obvious how to extend those methods to handle DRCs for advanced technology nodes. Previous machine learning approaches on DRC focus on predicting DRC from early design states, e.g [5] proposes to predict final DRC from macro floorplan.

3 Method

Placement: Given a set of PMOS and NMOS devices, the goal of placement is to place them on the PMOS row and NMOS row of the cell layout while satisfying technology constraints. In addition to device placement, cell pin locations should also be specified during placement.

Previous placers often separate placement into two steps: pairing and ordering. The pairing step pairs up each PMOS device with a NMOS device to form device pairs. The ordering step generates placement order of device pairs. And the final placement can be inferred from placement order and pairs. These two steps are interdependent, so solving one after the other is sub-optimal. Therefore we design a simulated annealing based algorithm that does both pairing and ordering simultaneously.

Simulated annealing makes *moves* on a placement representation which specify the placement order of pins, ordering of NMOS and PMOS devices, and whether to flip a device orientation (switching the source and drain positions). It optimizes a scoring function which is a weighted sum of cell width, congestion estimation and technology constraint violations. These *moves* can be categorized either by the types of moves or by the targeted devices of the moves. The *Flip* changes all targeted devices flip flag. The *Swap* swaps targeted devices. The *Move* moves targeted devices to a specific location. The target devices can be either consecutive PMOS devices, consecutive NMOS devices, a consecutive PMOS/NMOS device pair, or pins.

The simulated annealing algorithm is implemented based on the modified Lam annealing schedule [2] that requires no hyper-parameter tuning.

Routing has two steps: a genetic algorithm-based routing step and a RL-based DRC fixing step. The genetic algorithm drives a maze router to create many routing candidates, and the DRC RL agent reduces the number of DRCs of a given routing candidate.

The DRC RL agent only fixes M1 DRC errors. M1 is the lowest routing layer with most difficult DRC issues. Other DRC errors can be easily pruned out during maze routing. The RL *game* is to incrementally add additional M1 routing segments in order to reduce M1 DRCs. The observation space of the game includes the routes in M1, the DRC positions, and routing mask. The action space is the M1 grid that will be routed next. The rewards include a small negative reward given at each step and a large positive reward associated with DRC reduction.

We use the Proximal Policy Optimization (PPO) [15] algorithm to implement the RL agent. To build the policy and value networks for PPO, we need to consider two requirements: first it should be invariant to the number of nets, and second it should be invariant to the cell width, i.e. W^{M1} . Cell height H^{M1} is a constant for a given library. Our network design is shown in Figure 1.

The generic algorithm based routing algorithm uses routing segments as the genetic representation in that it ensures that good *routing islands* in the routing structure are preserved during genetic operations such as crossover and mutation. The fitness of each individual chromosome in a generation is evaluated based on two metrics: the number of unrouted terminal pairs and the number of DRCs.

Algorithm 1 NVCell Routing Flow

Input: nets \mathbb{N} , net terminals \mathbb{T}_n , generations G , population K

Output: DRC free routing candidates \mathbb{R}

create the terminal pair set \mathbb{P}

Maze route initial routing candidates $\{\mathbb{R}_1, \dots, \mathbb{R}_k\}$ for the first generation with random order

for $g \leftarrow 1$ **to** G **do**

for $i \leftarrow 1$ **to** K **do**

 select \mathbb{R}_{dad} and \mathbb{R}_{mom} from $\{\mathbb{R}_1, \dots, \mathbb{R}_k\}$

$\mathbb{R}' \leftarrow crossover(\mathbb{R}_{dad}, \mathbb{R}_{mom})$

$\mathbb{R}' \leftarrow mutate(\mathbb{R}')$

$\mathbb{R}'_i \leftarrow$ Maze route unrouted pairs with random order

 run RL DRC fixer on route \mathbb{R}'_i

$DRC(\mathbb{R}'_i) \leftarrow$ remaining DRCs

if $DRC(\mathbb{R}'_i) == 0$ **then**

 return \mathbb{R}'_i

$\{\mathbb{R}_1, \dots, \mathbb{R}_k\} \leftarrow$ Top K fitness from $\{\mathbb{R}'_1, \dots, \mathbb{R}'_k, \mathbb{R}_1, \dots, \mathbb{R}_k\}$

Other metrics can be also added into fitness function, such as total wiring cost or Design For Manufacturing (DFM) metrics. The complete routing flow is shown in Algorithm 1.

4 Results

NVCell is implemented with Python and runs on a server with multiple Intel Xeon CPUs where a maximum of 20 threads are used in the implementation. It generates a simplified grid-based cell layout, which is given to a separate Perl program called Sticks which handles DRC checking and conversion to tapeout quality Cadence Virtuoso layout. **NVCell** produces LVS/DRC clean cells for over 75% of a total of over 1000 single-row standard cells on an industrial standard cell library (multi-row cells are not currently supported by our Sticks program). Even more cells can be successfully generated if we relax the width optimization during placement. 22% of these generated cells have shorter width (between 1 to 8 poly columns) than the same cells in the library, and the rest have the same width. Although not all the cells in the library are designed to have minimum width, we did find meaningful size reduction on some critical arithmetic logic cells and flip-flops. This result shows **NVCell** can produce competitive cell layout as human designer.

For runtime, most of cells run within seconds to a few minutes. The largest cells finish in a couple of hours. We expect to significantly reduce the runtime when the implementations of **NVCell** and Sticks are switched from Python/Perl to C++.

The DRC RL agent is implemented based on stable-baselines [4] PPO2 (GPU-enabled implementation of PPO) algorithm. The RL environment is implemented based on the OpenAI GYM [1] framework. The training is conducted on a NVIDIA V100 GPU. We train the agent based on a 19-poly wide flip-flop cell. The environment is initialized with maze routing of all the nets within the cell based on a random net order. Note that PPO2 algorithm runs multiple environments in parallel, and each environment resets with a new random net order, so the agent will see many different initial routes and different DRC errors.

Although trained on only one cell, the model is actually transferable to all the cells we experimented with. Figure 2 shows the Stick diagram of both initial routes and routes after DRC fixing for a 35-poly wide specialty flip-flop. The DRC violations, indicated by dotted red lines on Figure 2(a), only appear between tracks that are within a certain distance, forming local patterns. These local DRC patterns can therefore be identified by the convolution networks within the RL agent to produce the policy. Because we train the model with many random routes, even only trained on one cell, the agent already sees many different routing configurations, which results in many DRC patterns. These DRC patterns are also invariant between different regions in the same cell or different cells and is why the trained model is transferable to new cells not seen during training.

5 Conclusions

We demonstrate that RL can be leveraged to fix standard cell routing DRCs incrementally from existing routes in an advanced technology node. Together with a genetic algorithm and simulated annealing, we build a standard cell layout generator **NVCell** that can generate competitive layouts for a majority of cells in a standard cell library.

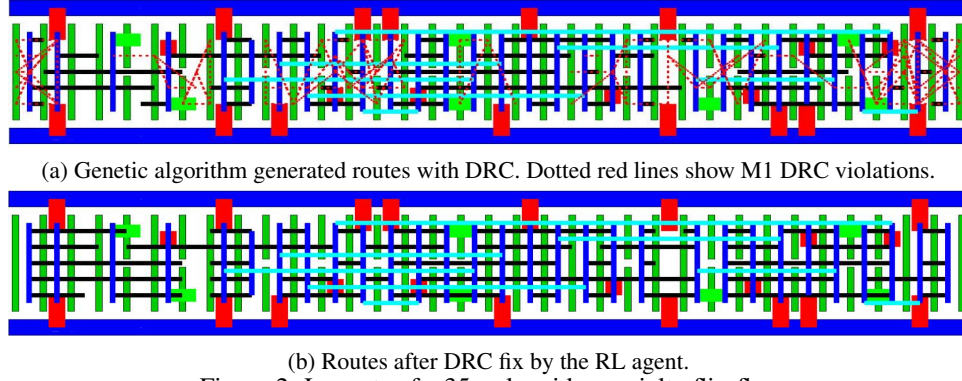


Figure 2: Layouts of a 35-poly wide specialty flip-flop

References

- [1] G. Brockman and et al. Openai gym. *CoRR*, abs/1606.01540, 2016. URL <http://arxiv.org/abs/1606.01540>.
- [2] V. A. Cicirello. On the design of an adaptive simulated annealing algorithm. In *International Conference on Principles and Practice of Constraint Programming First Workshop on Autonomous Search*, 2007.
- [3] Y. He and F. S. Bao. Circuit Routing Using Monte Carlo Tree Search and Deep Neural Networks. *arXiv e-prints*, June 2020.
- [4] A. Hill and et al. Stable baselines. <https://github.com/hill-a/stable-baselines>, 2018.
- [5] Y. Huang, Z. Xie, G. Fang, T. Yu, H. Ren, S. Fang, Y. Chen, and J. Hu. Routability-driven macro placement with embedded cnn-based prediction model. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 180–185, 2019. doi: 10.23919/DATE.2019.8715126.
- [6] K. Jo, S. Ahn, T. Kim, and K. Choi. Cohesive techniques for cell layout optimization supporting 2d metal-1 routing completion. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 500–506, 2018.
- [7] K. Jo, S. Ahn, T. Kim, and K. Choi. Cohesive techniques for cell layout optimization supporting 2d metal-1 routing completion. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 500–506, 2018.
- [8] Y. Li and et al. Ntucell: A dda-aware cell library generator for finfet structure with implicitly adjustable grid map. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.
- [9] H. Liao and et al. A Deep Reinforcement Learning Approach for Global Routing. *Journal of Mechanical Design*, 142(6), 11 2019.
- [10] H. Liao and et al. Attention Routing: track-assignment detailed routing using attention-based reinforcement learning. *arXiv e-prints*, Apr. 2020.
- [11] A. Lu, H. Lu, E. Jang, Y. Lin, C. Hung, C. Chuang, and R. Lin. Simultaneous transistor pairing and placement for cmos standard cells. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1647–1652, 2015.
- [12] A. Mirhoseini and et al. Chip placement with deep reinforcement learning. 2020. URL <https://arxiv.org/abs/2004.10746>.
- [13] V. Mnih and et al. Playing atari with deep reinforcement learning. 2013. URL <http://arxiv.org/abs/1312.5602>.
- [14] N. Ryzhenko and S. Burns. Standard cell routing via boolean satisfiability. In *DAC Design Automation Conference 2012*, pages 603–612, 2012.
- [15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL <http://arxiv.org/abs/1707.06347>.
- [16] N. A. Sherwani. *Algorithms for VLSI Physical Design Automation*. Kluwer Academic Publishers, USA, 3rd edition, 1999.
- [17] P. Van Cleeff, S. Hougardy, J. Silvanus, and T. Werner. Bonncell: Automatic cell layout in the 7nm era. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2019.
- [18] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, volume 8, pages 229–256, 1992. doi: 10.1007/BF00992696.
- [19] S. Wimer, R. Y. Pinter, and J. A. Feldman. Optimal chaining of cmos transistors in a functional cell. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(5):795–801, 1987.