Ph.D. DISSERTATION

Synthesis and Optimization of Standard Cells and Design Quality Prediction in Physical Design Automation

물리적 설계 자동화에서 표준셀 합성 및 최적화와 설계 품질 예측 방법론

BY

BAEK KYEONGHYEON

FEBRUARY 2023

DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Ph.D. DISSERTATION

Synthesis and Optimization of Standard Cells and Design Quality Prediction in Physical Design Automation

물리적 설계 자동화에서 표준셀 합성 및 최적화와 설계 품질 예측 방법론

BY

BAEK KYEONGHYEON

FEBRUARY 2023

DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Synthesis and Optimization of Standard Cells and Design Quality Prediction in Physical Design Automation

물리적 설계 자동화에서 표준셀 합성 및 최적화와 설계 품질 예측 방법론

지도교수 김 태 환

이 논문을 공학박사 학위논문으로 제출함

2022년 12월

서울대학교 대학원

전기·정보 공학부

백 경 현

백경현의 공학박사 학위 논문을 인준함

2022년 12월

위 원 장: _____

부위원장: _____

위 원: _____

위 원: _____

위 원: _____

Abstract

In the physical design of chip implementation, designing high-quality standard cell layout and accurately predicting post-route DRV (design rule violation) at an early stage is an important problem, especially in advanced technology nodes. This dissertation presents two methodologies that can contribute to improving the design quality and design turnaround time of physical design flow.

Firstly, we propose an integrated approach to the two problems of transistor folding and placement in standard cell layout synthesis. Precisely, we propose a globally optimal algorithm of search tree based design space exploration, devising a set of effective speeding up techniques as well as dynamic programming based fast cost computation. In addition, our algorithm incorporates the minimum oxide diffusion jog constraint, which closely relies on both of transistor folding and placement. Through experiments with the transistor netlists and design rules in advanced node, our proposed method is able to synthesize fully routable cell layouts of minimal size within a very fast time for each netlist, outperforming the cell layout quality in the manual design.

Secondly, we propose a novel ML based DRC hotspot prediction technique, which is able to accurately capture the combined impact of pin accessibility and routing congestion on DRC hotspots. Precisely, we devise a graph, called *pin proximity graph*, that effectively models the spatial information on cell I/O pins and the information on pin-to-pin disturbance relation. Then, we propose a new ML model, which tightly combines GNN (graph neural network) and U-net in a way that GNN is used to embed pin accessibility information abstracted from our pin proximity graph while U-net is used to extract routing congestion information from grid-based features. Through experiments with a set of benchmark designs using advanced node, our model outperforms the existing ML models on all benchmark designs within the fast inference time in comparison with that of the state-of-the-art techniques.

keywords: standard cell, transistor placement, transistor folding, design rule violation, pin accessibility
student number: 2017-21739

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Standard Cell Layout Synthesis

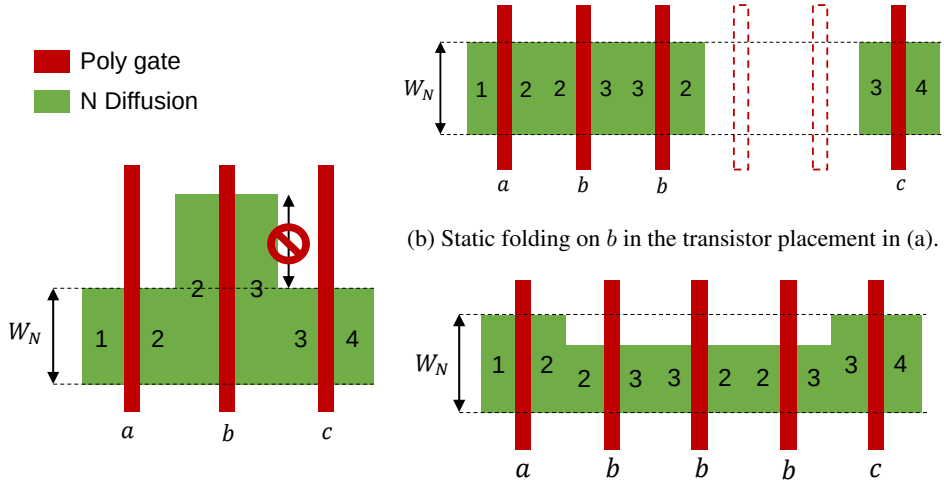
Standard cell layout synthesis belongs to a constrained optimization problem whose most important objective is to find a solution that minimizes the cell area. Since cell layout optimization is NP-hard [6], it is generally performed in a sequence of three steps: transistor folding, transistor placement, and in-cell routing, in which *transistor placement* plays a key role in minimizing cell area. Through transistor placement, cell of minimum area, regardless of in-cell routability, can be obtained by ordering transistors in a way to maximize the possibility of diffusion sharing between two transistors adjacent to each other. A lot of methods of transistor placement have been proposed. The existing methods can be classified as: integer linear programming (ILP) based (e.g., [7, 8]), graph theory based (e.g., [9, 10, 11, 12]), Boolean satisfiability (SAT) based (e.g., [13, 14, 15]), simulated annealing based (e.g., [16]), dynamic programming based (e.g., [17]), machine learning based (e.g., [18, 19]), and search tree exploration based (e.g., [20, 21]). Also, numerous in-cell routing methodology for synthesizing optimized standard cell layout has been proposed. (e.g., [22, 23, 24, 25, 26, 27, 28])

In practical cell layout synthesis, the size of each transistor is determined individually by taking into account the cell's performance constraints. For a pre-specified limit

on the width, denoted by W_P and W_N, of P and N diffusions, transistors of large size cannot be implemented with single channels. *Transistor folding* is the process of splitting a large transistor into multiple small and equal-sized (sub)-transistors, called *fingers*, that are connected in parallel and placed contiguously with diffusion sharing [29]. *Static folding* refers to folding every transistor in the cell with its minimal number of fingers such that their channel widths[1] exactly equal to the value of W_P or W_N while *dynamic folding* refers to folding the transistors individually with some number of fingers of uniform channel width that is less than or equals to the value of W_P or W_N. Fig. 1.1(a) illustrates no folding on transistor b in a transistor placement, resulting in the violation of W_N constraint in b. On the other hand, Fig. 1.1(b) shows a static folding on transistor b, resolving the violation, but requiring a diffusion break between transistors b and c. Fig. 1.1(c) shows a dynamic folding on b. It creates one more fingers ($2 \rightarrow 3$) over that in Fig. 1.1(b), but enables a diffusion sharing between b and c. Note that dynamic folding is a generalization of static folding. Given W_P and W_N, the cell size is determined by the following closely inter-related factors: (1) the number and orientation (i.e., flip or no-flip) of fingers for each transistor, (2) the placement of fingers, and (3) the amount of diffusion sharing.

To our knowledge, all existing works have addressed transistor placement with no or partial consideration of transistor folding (e.g., [7, 8, 10, 11, 12, 13, 17, 18, 19, 21, 29]). Jo *et al.* [12] expressed input netlist as a graph model where each node represents a distinct net and an edge exists between two nodes if there is a transistor connecting the nets corresponding to the nodes, and obtained a transistor placement that maximizes the number of diffusion sharing through searching Euler trail on the graph; Iizuka *et al.* [13] transformed the problem of finding a minimum-width multi-row transistor placement into a pseudo-Boolean optimization problem, and made use of SAT solver to discover optimal placements; Li *et al.* [17] utilized a dynamic programming based algorithm to find an optimal transistor placement with the objective

[1]The channel width means to the size of the finger.



(a) No folding on transistor b in a transistor placement.

(b) Static folding on b in the transistor placement in (a).

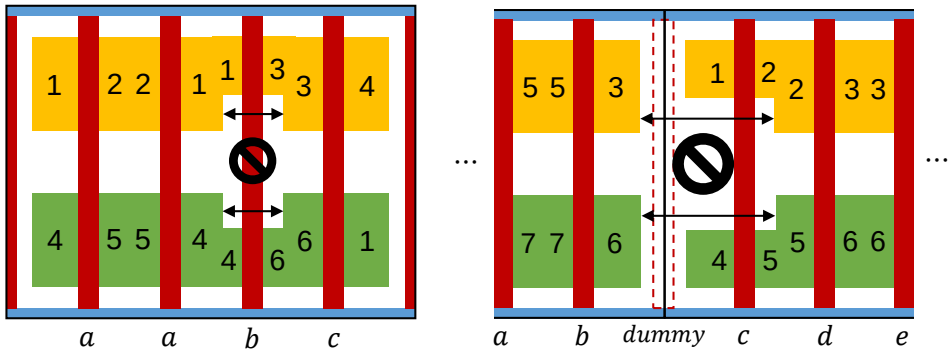
(c) Dynamic folding on b in the transistor placement in (a).

Figure 1.1: An illustration of transistor folding on a transistor and the impact of folding on diffusion sharing. (W_N represents the maximum width constraint of N diffusion.)

of not only minimizing cell area but also maximizing the in-cell routability of the transistor placement; Lee *et al.* [18] tried to search a minimum-area transistor placement using Monte-Carlo search tree algorithm and estimated the ranking of the cell delay of the placements by exploiting convolutional neural network (CNN) model; Ren *et al.* [19] proposed a reinforcement learning based transistor placement engine to find optimal results. For a set of pMOS and nMOS transistor sizes in a cell, Kim and Kang [29] proposed a polynomial-time algorithm of finding an optimal folding size, equivalently optimal values of W_P and W_N, which leads to minimize the resulting cell size. The algorithm has two disabilities. It did not consider the diffusion breaks which may be caused by the transistor placement and finger orientation. Further, it assumed to produce all fingers of identical size. Gupta and Hayes [7] formulated the combined problem of transistor folding and placement as a 0-1 integer linear programming (ILP) problem to produce optimal results. However, in a strict sense, they attempted to solve the problem of, so-called *transistor placement with static folding* rather than to solve

the problem of *transistor placement with dynamic folding*, in that they fixed the number of fingers to $\lceil \frac{t_P}{W_P} \rceil$ or $\lceil \frac{t_N}{W_N} \rceil$ for each pMOS or nMOS transistor where t_P and t_N respectively represent the size of the pMOS or nMOS transistor, and then determined their position and orientation to minimize the cell size. Lu *et al.* [8] proposed an ILP based approach to the problem of transistor placement combined with sizing of individual fingers to minimize the resulting cell area. However, the approach assumed that the number of fingers for each transistor has been given as input and never considered as a parameter to be optimized. To sum up, the works in [7, 8, 11, 12, 13, 17, 18, 19, 29] did not fully exploit the various effects of dynamic folding on diffusion sharing, ultimately on reducing the cell size.

On the other hand, Cortadella [10] constructed a graph model from the transistor netlist of an input cell to formulate an instance of dynamic folding problem, and solved the folding problem by minimally adding new edges to the graph until the existence of Eulerian path in the graph is guaranteed. He applied a mixed ILP formulation to find the minimal number of extra edges called *Eulerization cost*, which was used to estimate the number of diffusion breaks in transistor placement. Since the method processed two graphs independently, one for netlist of nMOS transistors and one for pMOS, the Eulerization costs totally ignored the gate sharing of transistors, also known as *transistor pairing*, in placement. Recently, Cleef *et al.* [21] proposed a search tree based exploration approach to find the best placement and dynamic folding of transistors. As a search tree pruning technique, they estimated the number of diffusion breaks by using a graph model similar to that in [10]. They performed pMOS transistor placement and nMOS transistor placement separately and combined the two placement solutions for generating entire placement. The fundamental limitation of performing pMOS and nMOS placements separately is that it incurs many *gate (vertical) misalignments*. Gate misalignment incurs when a pair of pMOS and nMOS transistors sharing a net are not aligned vertically in a single column. Since those two transistors cannot be implemented with a vertical straight poly gate, a metal resource



(a) OD jog violation occurs when synthesizing cell layout.

(b) OD jog violation occurs when abutting cells in cell placement.

Figure 1.2: Design rule for minimum OD (oxide-diffusion) jog.

is required to connect them, which eventually hinders in-cell routability.

Recently, a number of transistor placement methods using SMT (Satisfiability Modulo Theories) solvers with OMT (Optimization Modulo Theories) are proposed. (e.g. [2, 15]) Lee *et al.* [15] constructed an SMT-based framework called SP&R for performing simultaneous transistor placement and in-cell routing. Jo and Kim [2] proposed an optimal transistor placement method combined with global in-cell routing to ensure good routability of the placement solutions. They defined the global routing bin and tried to find the solutions with the objectives of minimizing a weighted sum of cell area and global routing wirelength. One critical limitation of the SMT-based methods is that as the cell size increases, the runtime increases exponentially. Thus, practically, it is not acceptable to integrate dynamic folding option in their SMT-based placement formulations.

Clearly, it is highly desirable to solve the two problems of transistor folding and placement in an integrated fashion. However, all prior works have not proposed fully integrated solutions. One of the main reasons is an exponential time complexity on exploring the search space. Nevertheless, as the technology scaling continues, solving the two problems in an integrated framework is appealing since it introduces new DRs (design rules), which are closely affected by both of transistor folding and place-

ment. One such noticeable DR is the minimum OD (oxide diffusion) jog rule [30]: Transistors can have different OD region heights according to dynamic folding. When transistors in a cell with different OD heights abut or cells with different OD heights in the cell boundaries abut, OD jog violation can occur as shown in Figs. 1.2(a) and (b).

1.2 Machine Learning for Electronic Design Automation

Electronic design automation (EDA) is one of the important research areas in electronic engineering. As the technology nodes shrink, the complexity of chip implementation flow exponentially increases. Currently, most EDA problems are known as NP-Complete problems, so it is required to develop algorithms that can effectively handle explosively large design spaces.

Machine learning (ML) plays an important role in our life these days. With the rapid growth of hardware computing power such as GPUs, machine learning has shown great performance on classification, detection, and design space exploration problems in recent years. Commercial EDA tools adopt algorithmic approach for designing chips, but it has not solved the scalability problems caused by the rapid increase in the size of design space on modern IC designs. Machine learning-based approaches can be alternative solutions for EDA problems, so in recent years, machine learning for EDA has gained popularity and lots of studies have been proposed. These works tried to efficiently exploit machine learning algorithms on the current EDA tool chain, and covered the whole process of chip implementation flow composed of logic synthesis, placement, clock tree synthesis, routing, timing analysis, and manufacturing.

Current works can be classified as three categories; *acceleration of existing algorithms*, *early stage quality prediction* and *design space exploration/optimization*. The detailed explanations of each category are as follows.

The acceleration of the existing algorithm is enabled by maximizing the parallel

computing using GPU programming systems and AI SW libraries. It covers global placement [31] and global routing [32, 33]. These approaches do not use machine learning itself, but it modifies the existing algorithmic approaches to maximize parallelism. DREAMPLACE [31] started with the idea that the backpropagation of deep learning and the cost optimization process of global placement have similar computational methods, achieving substantial speedup compared to the conventional approaches. FastGR [32] and GAMER [33] also utilize the potential of GPUs on global routing. These works propose a pathfinding-level parallel 3D routing methodology implemented on the CUDA platform.

Fast and accurate early-stage quality prediction is essential to chip implementation since it can boost the total design time. Supervised learning is exploited in this category, training the model by extracting information from previously implemented designs, and then predict the quality of the new design based on this. Baek *et al.* [34] and Lu *et al.* [35] utilized graph neural network to predict post-route DRC hotspots and timing information in terms of total negative slack at the placement stage of the physical design flow. MAVIREC [36] was inspired by U-Net architecture [4] adopting three-dimensional convolutional layers to perform IR drop estimation in early stages. GRANNITE [37] learned the toggle rate of combinational logic from the RTL simulation results, enabling fast power estimation.

Design space exploration/optimization is one of the most important points in solving the EDA problems, which considers how to handle the large design space and find the optimization points in a fast time. Recently, bayesian optimization (BO) and reinforcement learning (RL) techniques have shown high performance in design space optimization problems. A lot of studies have been proposed to utilize these techniques on chip implementation flow including standard cell layout synthesis [19], logic synthesis [38], macro placement [39], routing [40, 41] and P&R flow optimization [42, 43]. NVcell [19] proposed a methodology to fix the routing DRCs using reinforcement learning-based agents from the initial routing solutions performed by genetic algo-

rithm. DRiLLS [38] tried to find the optimal logic synthesis command order of the commercial tool to minimize the area and delay of the netlist. Mirhoseini *et al.* [39] proposed an RL agent that extracts features from netlist graphs using the GNN model and optimizes the PPA of chips through efficient macro block placement. Liao *et al.* [40, 41] suggested deep Q network (DQN) [44] based global router and attention-based track assignment algorithm in detailed routing. RL-sizer [42] utilized deep deterministic policy gradient (DDPG) algorithm [45] to find the best transistor sizing for timing optimization. Agnesina *et al.* [43] proposed a framework that finds the optimal placement parameters of the P&R flow with only a small number of inferences.

1.3 Prediction of Design Rule Violation

In modern integrated circuit designs, the introduction of new complex design rules at the advanced technology nodes makes implementing chips very challenging. Particularly, the complex design rules put so much burden on physical design, demanding lots of iterations on the time-consuming process of cell placement and net routing to clean up all DRVs (design rule violations) before tapping out. Thus, at the placement stage, if we were able to identify, with high confidence, DRC (design rule check) hotspots that would be likely to occur at the routing stage, we can pay more attention on the cell placements in those DRC hotspots, so that the iteration process of placement and routing should quickly converge to DRV-clean.

Traditionally, commercial place-and-route tools have used *congestion map* delivered from initial global routing (GR) or trial routing as an early DRC hotspot estimator at the placement. However, as the technology node shrinks, the DRC hotspots predicted by GR congestion map alone is not fully correlated with the post-route DRC hotspots, as shown in Fig. 1.3. At the advanced node, it is known that *I/O pin inaccessibility* in standard cells is one of the main causes of the DRC hotspot misprediction. Since the high increase of pin inaccessibility in routing is attributed by the cell size re-

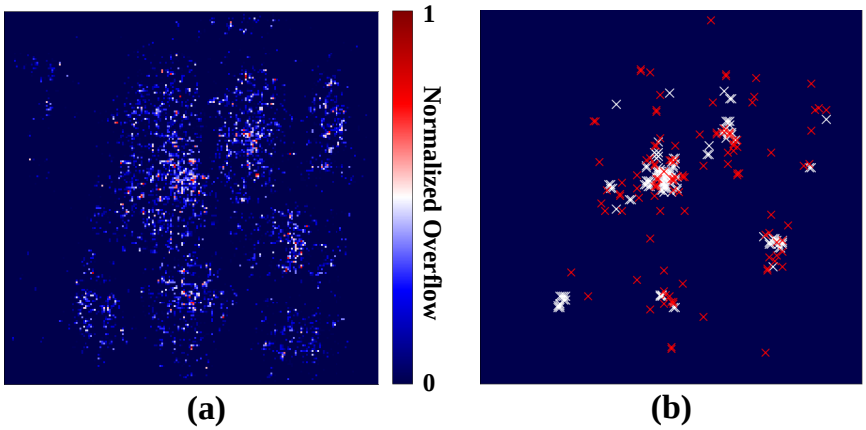


Figure 1.3: Comparison between GR congestion map and actual DRC hotspots on ECG circuit. (a) GR congestion. (b) An overlay of actual hotspots (white crosses) and predicted hotspots (red crosses) by GR congestion in (a).

duction and the reduced number of routing tracks over the cells, one of the top priority in placement stage is resolving pin inaccessibility issue [46, 47, 48, 49]. Furthermore, as the routing complexity considering complex design rules increases, routers expose more often unpredictable behaviors, which is also one of main causes of the increase of DRC hotspot misprediction.

To sum up, *routing congestion* and *pin accessibility* are two major causes of DRVs. The prior works of DRC hotspot prediction previously had considered routing congestion, but paid little or no attention on pin accessibility, then gradually taking into account pin accessibility as the technology node shrinks.

The shortage of routing capacity in the area of routing congested region ends up committing design rule violations. In order to discover the routing congestion related DRVs, Chan *et al.* [50, 51] extracted, as features, the number of pins, the incoming and outgoing hyper-edges, and others from a local window, and used a SVM (support vector machine) model to predict occurrence of DRVs in the window. Zeng *et al.* [52] proposed random forest-based prediction model, and utilizes SHAP tree explainer to analyze the impact of input features on output prediction. Hung *et al.* [53] used a

CNN (convolutional neural network) model to capture routing congestion around a local window based on the congestion map (extracted from global routing information) of adjacent grids. Yu *et al.* [54] exploited cGAN(conditional generative adversarial nets) model to generate routing congestion map image on FPGA circuit. Xie *et al.* (RouteNet) [55] utilized FCN (fully convolutional network) structure. They divided the entire placement region into grids of small size, and concatenated various DRV related features for each grid into three-dimensional feature map to maintain spatial information. One common feature of the aforementioned works is that they all focused on developing ML (machine learning) models of routing congestion aware DRC hotspot prediction and never incorporated pin accessibility into their models.

Recently, noticeable studies addressing pin accessibility aware DRC hotspot prediction have emerged. Yu *et al.* [56, 57] exploited CNN to predict DRVs mainly induced by low pin accessibility in a local window. They used fine-grained pin pattern images as input feature to express the shape of the pins in detail. Liang *et al.* [58] suggested a novel neural networks structure called J-net, which is a sort of modified model of U-net, adding a series of down-sampling modules in front of the encoding path of U-net to accommodate not only high-resolution pin pattern images but also grid-based feature maps as input feature. Although Yu *et al.* [56, 57] and J-net [58] both have tried to express pin information with fine-grained pin pattern images, these works have two critical limitations.

Firstly, local pin accessibility cannot be accurately modeled by pin pattern image alone, which represents only the detailed shape and location of the pins. Rather, DRC hotspots heavily rely on how metal wires approach to their target pins and how many metal wires go through over the pins together with the situation of accessing neighboring pins even though the pin information such as shape or location are identical. For example, Fig. 1.4 illustrates the different pin accessing situations with the same pin layouts, in which pins p_1, p_2, and p_3 are easily accessible if nothing tries to obstruct the connections as shown in Fig. 1.4(a). However, as illustrated in Fig. 1.4(b), if other

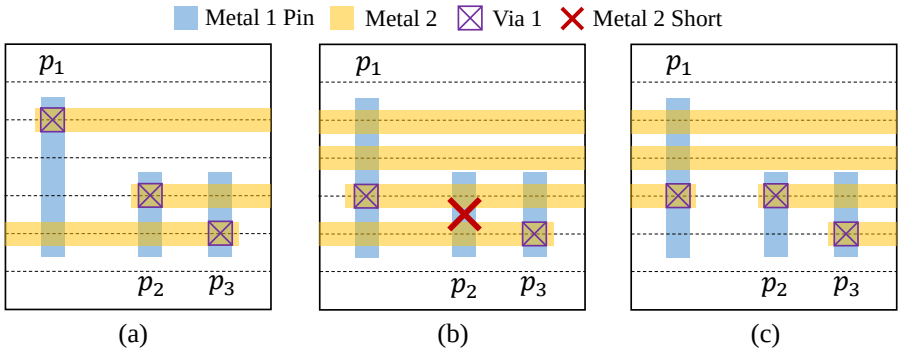


Figure 1.4: Pin accessing situations. (a) The three pins are accessed easily if any metal wires pass on them. (b) p_2 is blocked when metal wires obstruct the pins. (c) p_2 can be accessible if p_1 and p_3 are accessed through different directions even in the same situations with (b).

metal wires expand metal tracks near the pins, a DRV occurs since p_2 is not accessible by being blocked by other metals. Even with the same pin layouts, if p_1 and p_3 are accessed through different directions, the inaccessibility to p_2 can be resolved as shown in Fig. 1.4(c). Secondly, using high-resolution pin pattern images incur significant additional run-time as well as memory overhead to the prediction models. J-net [58] used pin pattern images which required more than thousands of pixels for representing pin layouts in a single grid. However, as mentioned above, the pin layout itself does not accurately describe the pin accessibility problem. Furthermore, the models consume most of the computation time for processing the massive size of images.

1.4 Contributions of This Dissertation

In this dissertation, we present several methodologies to improve design quality and design turnaround time in physical design stage.

In Chapter 2, we propose a methodology of synthesizing standard cell layout of advanced nodes under 10nm process with simultaneous transistor folding and place-

ment [59]. This work overcomes the critical limitation of existing transistor placement works, that is, not fully integrating transistor folding into transistor placement. The contributions of our work can be summarized as:

1. We *propose a search tree based algorithm that exhaustively explores the solution space of transistor folding and placement*. Unlike the conventional methods targeting to generate exactly one compact (optimal) layout for each input netlist of transistors in a cell, our algorithm is able to produce diverse optimal layouts, so that designers can choose the layout that is best suited to the implementation objectives and constraints.

2. To be computationally viable, we *devise a set of well-defined speeding up techniques as well as dynamic programming based fast cost computation* to effectively and efficiently prune the search space of transistor folding and placement without losing optimality.

3. We take into account the minimum OD jog constraint in our integrated algorithm of transistor folding and placement, so that *OD jog violation never occurs inside the synthesized cells.* We show in experiments that tightly linking the two tasks of transistor folding and placement to avoid OD jog violation can save more cell area, which otherwise, were necessary to resolve OD jog violations afterwards.

4. We provide a fast routability estimation metric to assess transistor placement solutions and a cell layout synthesis flow to explore cell layouts by varying the cell size constraint. With using the routability estimator and full synthesis flow down to in-cell routing, We can generate complete cell layout solutions of small-area with effective use of metal resources in in-cell routing.

In Chapter 3, we propose a novel methodology of predicting DRC hotspots with a combined model of GNN (graph neural network) and U-net to accurately and efficiently capture the compound impact of pin accessibility and routing congestion on

DRC hotspot prediction [34]. The main contributions of our work can be summarized as follows:

1. We propose a *pin proximity graph* that effectively models not only the spatial information of each pin but also the information on how each net accesses to its target pin. To the best of our knowledge, this is the first work that uses a graph for modeling pin information.

2. We propose *GNN architecture compatible with pin proximity graph*. The suggested architecture aggregates the information of the neighboring pins from the reference pin to consciously formulate pin accessibility.

3. We propose a novel deep ML model, called PGNN (Pin accessibility aware GNN and U-Net), which is a combined model of GNN and U-net. PGNN can adopt pin proximity graph as well as grid-based feature map as input feature. GNN in PGNN embeds pin accessibility information of each grid taken from the pin proximity graph, and U-net in PGNN extracts routing congestion information from the grid-based features.

4. In comparison with the prior works, which have utilized high-quality pin shape images for the construction of pin accessibility metrics, our graph formulation of PGNN tremendously saves the runtime for both training and inference.

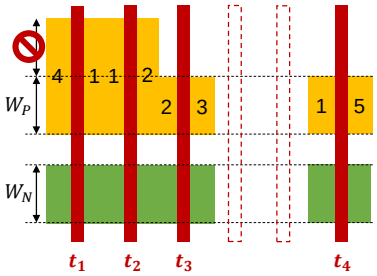
Chapter 2

Standard Cell Layout Synthesis of Advanced Nodes with Simultaneous Transistor Folding and Placement

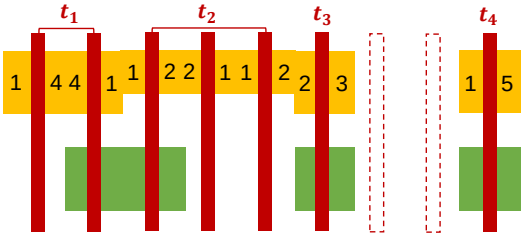
2.1 Motivations

If transistor folding is performed before or after transistor placement, optimal solutions may not be found. Fig 2.1 shows the impact of transistor folding on the cell width (i.e., cell size). Fig 2.1(a) shows an optimal transistor placement for four transistor pairs t_1, t_2, t_3, and t_4 with no transistor folding, which requires total of 6 poly gates including 2 dummy gates to make a diffusion break. Note that for transistor pairs t_1 and t_2, each has P diffusion region of width larger than W_P. Thus, it is required that the transistors of t_1 and t_2 in P diffusion region should be folded. There will be three possible temporal combinations of transistor folding and placement:

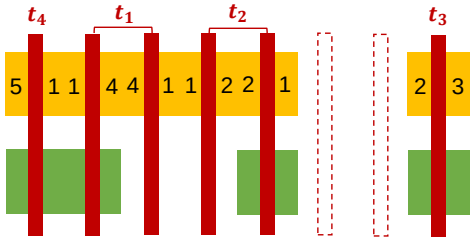
- **Combination 1** (*placement* $\rightarrow$ *folding*): performing transistor placement first and then performing transistor folding. For example, Fig. 2.1(b) shows an optimal folding result for the optimal transistor placement in Fig. 2.1(a), in which to avoid diffusion breaks between t_1 and t_2 and between t_2 and t_3, t_1 and t_2 are folded with two and three fingers, respectively. This results in a total of 9 poly gates including 2 dummy gates between t_3 and t_4.



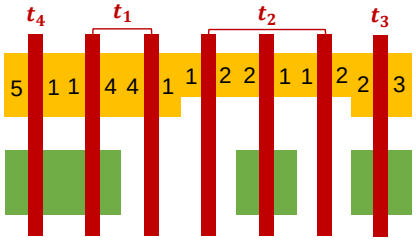
(a) Optimal placement with no consideration of folding.



(b) *Temporal combination 1*: Optimal placement and then optimal folding, allocating 9 poly gates.



(c) *Temporal combination 2*: Optimal folding and then optimal placement, allocating 8 poly gates.



(d) *Temporal combination 3*: Simultaneous optimal folding and placement, allocating 7 poly gates.

Figure 2.1: An example illustrating three possible temporal combinations of transistor folding and placement.

- **Combination 2** (*folding* $\rightarrow$ *placement*): performing transistor folding first and then performing transistor placement. For example, Fig. 2.1(c) shows an optimal placement result for the minimal number of fingers produced by applying folding individually to the transistor pairs in Fig. 2.1(a). This results in total of 8 poly gates including 2 dummy gates. Most of the existing works belong to this temporal combination (e.g., [7, 12, 13, 8, 15, 2]).

- **Combination 3** (*folding* + *placement*): performing transistor folding and placement simultaneously. For example, Fig. 2.1(d) shows an optimal result of folding and placement for the four transistor pairs in Fig. 2.1(a). Unlike *Combination 2*, t_2 is folded by three fingers to implement diffusion sharing with t_1 and t_3, saving two and one poly gates in comparison with the results produced by *Combinations* 1 and 2, respectively.

As illustrated in Fig. 2.1, it is highly desirable to perform two tasks of transistor folding and placement in an integrated framework, that is, *Combination 3* to produce cell layouts of minimal area.

2.2 Algorithm for Standard Cell Layout Synthesis

2.2.1 Problem Definition

Let t_i = (t_i^p and t_i^n) be a pair of pMOS and nMOS transistors sharing a gate poly in an input transistor netlist of CMOS standard cell. We assume a transistor pair set $\mathcal{T}$ is composed of such K pairs of transistors. Each pMOS or nMOS transistor t_i^x, $x = p$ or n is characterized by a tuple $(n_s, n_g, n_d, size, type)$ where n_s, n_g, and n_d represent source, poly gate, and drain nets of t_i^x, respectively, $size(t_i^x)$ denotes the transistor size[1], and $type \in \{pMOS, nMOS\}$ indicates the transistor type. Design rules to be satisfied in the process of transistor folding and placement are the followings. (Constant parameters are shown in parentheses.)

[1]We set $size(t_i^x)$ to the number of fins required in implementation.

- **DR 1** (*Diffusion break constraint* (N_{dummy}^{gate})): If two transistors adjoining each other have different diffusion nets in between them, at least N_{dummy}^{gate} number of dummy poly gates should be inserted between the transistors. Typically, the value of N_{dummy}^{gate} is 1 or 2 depending on the process technology used.

- **DR 2** (*Diffusion width constraint* (W_P, W_N, W_{MIN})): Diffusion widths, in terms of the number of fins, of all transistors should be larger than or equal to the value of W_{MIN} but should not exceed the value of W_P for pMOS or the value of W_N for nMOS.

- **DR 3** (*Oxide diffusion jog constraint* (L_{OD})): The oxide diffusion region height should be uniformly maintained. The uniformed height should be lengthened to be more than the value of L_{OD}.

Let $\Lambda(t_i, \cdot)$ be an instance of folding shape that can be produced by applying dynamic folding to both t_i^p and t_i^n to satisfy DR 2 i.e. the diffusion width constraint and DR 3 i.e. OD jog constraint. In addition, let S_i (= $\{\Lambda(t_i, 1), \Lambda(t_i, 2) \cdots\}$) be the set of all valid folding shapes of t_i and $len(\Lambda(t_i, \cdot))$ denote the horizontal length of folding shape $\Lambda(t_i, \cdot)$, expressed in terms of the number of poly gates. (Details on folding shapes and their generation will be described in Sec. 2.2.3.) Then, we define a function, called *folding function*, $\mathcal{F}(t_i)$, $i = 1, 2, \cdots, K$:

$$\mathcal{F}(t_i) = \Lambda(t_i, \cdot) \in S_i, \tag{2.1}$$

which maps transistor pair t_i to a legal folding shape in set S_i.

Problem 1 (*Transistor folding and placement*): *For an input transistor netlist of CMOS standard cell with K pairs of transistors, find (i) a mapping function $\mathcal{F}(\cdot)$ and (ii) a linear order of the folding shapes $\mathcal{F}(t_1), \mathcal{F}(t_2), \cdots, \mathcal{F}(t_K)$ which minimizes the quantity of $Cost$:*

$$Cost = \sum_{i=1,\cdots,K} len(\mathcal{F}(t_i)) + N_{dummy}^{gate} \cdot N_{break} - \alpha_{tot} \tag{2.2}$$

while satisfying DR 1, 2, and 3, where N_{break} is the number of diffusion breaks inserted between the folding shapes to meet DR 1 i.e. diffusion break constraint, and α_{tot} is the total number of vertical lines of poly gates that can be saved by abutting folding shapes[2].

2.2.2 Overall Flow

Fig. 2.2 shows the flow of our proposed cell layout generator, called CSyn-fp (Cell Synthesis with simultaneous Folding and Placement), which performs the following four steps: For an input transistor netlist $\mathcal{N}$ of cell with K transistor pairs and design rule parameters (N^{gate}_{dummy}, W_P, W_N, W_{MIN} and L_{OD}), (Step 1) *enumerating the set of all feasible folding shapes S_i* with no DR 2 and DR 3 violations for every transistor pair $t_i, i = 1, \cdots, K$ in $\mathcal{N}$; (Step 2) *building-up a search tree based design space exploration* to find mapping functions $\mathcal{F}(\cdot)$ and linear orders of the folding shapes $\mathcal{F}(t_1), \mathcal{F}(t_2), \cdots, \mathcal{F}(t_K)$ which minimize the quantity of $Cost$ in *Eq.2.2* while considering the impact of DR 1 and DR 3; (Step 3) *applying a conventional in-cell router* to the folding and placement results $\mathcal{L}_1, \cdots$ in Step 2 to produce legal and complete layouts $\mathcal{L}'_1, \cdots$ corresponding to $\mathcal{N}$. (Step 4) *splitting folding shapes* in solutions of Step 3 to further reduce cell area. (It is an optional step since it may induce additional in-cell routing resource.); (Step 5) If no legal cell layouts are found (all placement solutions are unroutable), reperform the algorithm by relaxing the minimum-area constraint to search routable placement solutions.

2.2.3 Step 1: Generation of Folding Shapes

We illustrate our method of folding shape generation for a transistor pair $t_i = (t^p_i, t^n_i) \in \mathcal{T}$ using an example shown in Fig. 2.3(a) where *size*$(t^p_i) = 6$, *size*$(t^n_i) = 4$, $W_P = 4$, $W_N = 3$, and $W_{MIN} = 2$.

Step 1.1 (*Generating all folding configurations for pMOS t^p_i*): This step generates all

[2]For example, see Fig. 2.5.

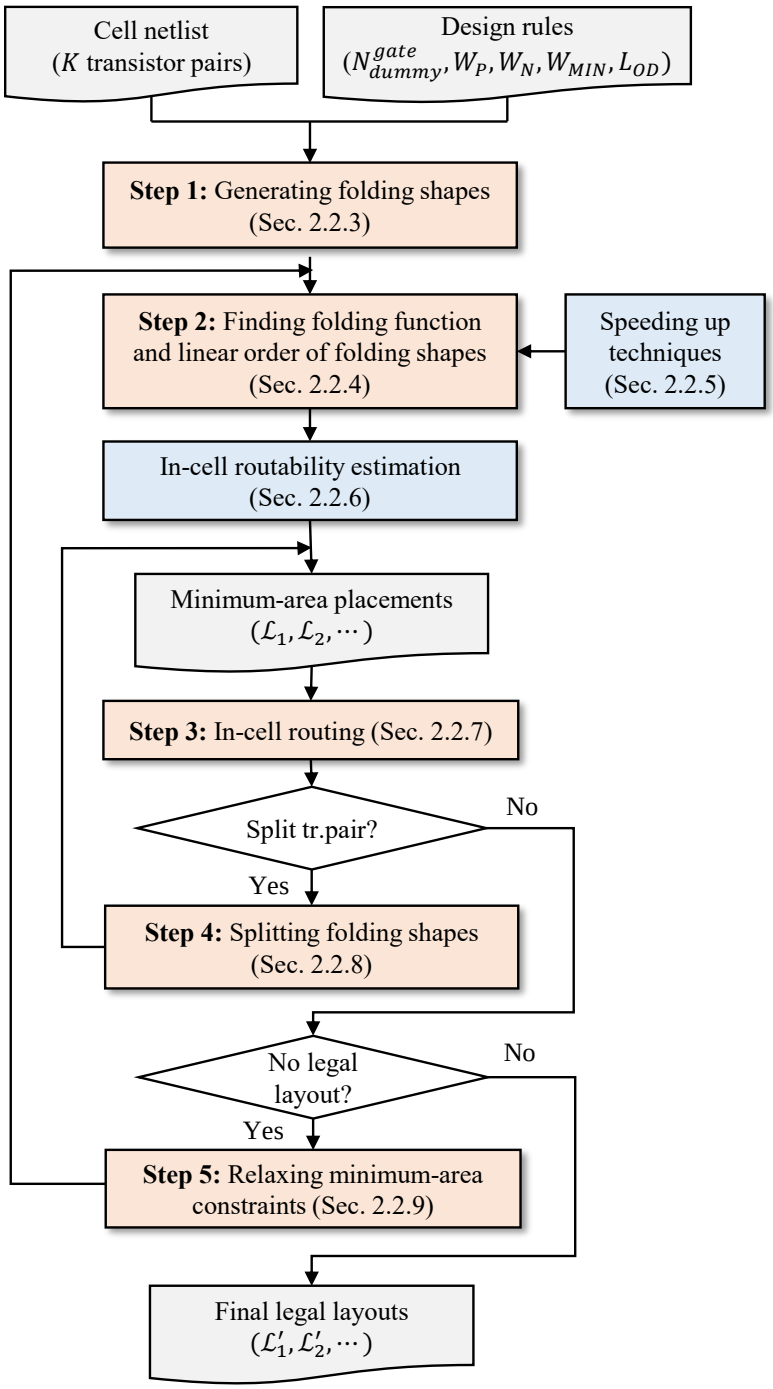


Figure 2.2: The flow of our proposed cell layout generator CSyn-fp.

feasible folding configurations for t_i^p while satisfying $W_P = 4$ and $W_{MIN} = 2$. For example, configuration Γ_1^p, shown in Fig. 2.3(b), is composed of two fingers, one with 4 fins and the other with 2 fins, thus total of 6 fins (= $size(t_i^p)$). Γ_2^p is produced by cyclic orientation of the diffusion nets by one-step move to the left or right while Γ_3^p and Γ_4^p are produced by applying left-to-right flipping to Γ_1^p and Γ_2^p, respectively.

Step 1.2 (*Generating all folding configurations for nMOS* t_i^n): This step is identical to step 1.1 except that the transistor to be folded is now t_i^n. For example, since $W_{MIN} = 2$, the bottom configuration shown in Fig. 2.3(c), which has a finger of size 1, is not allowed while Γ_1^n and Γ_2^n (orientation of Γ_1^n) are both feasible nMOS folding configurations.

Step 1.3 (*Generating all folding shapes* $\Lambda(t_i, \cdot) = (\Gamma_{j_1}^p, \Gamma_{j_2}^n, \delta)$): A distinct folding shape can be produced by a pair of pMOS and nMOS configurations and offset value (δ) with respect to the left alignment of the configurations. For example, for Γ_1^p and Γ_1^n, the minimum length folding shape is only that labeled as (Γ_1^p, Γ_1^n, 0) in Fig. 2.3(d) whereas for Γ_7^p and Γ_1^n, the minimum length folding shapes are the those labeled as (Γ_7^p, Γ_1^n, 0) (Γ_7^p, Γ_1^n, 1) in Fig. 2.3(d).

It should be noted that since some transistor pairs have the same values of pMOS size (i.e., the number of fins) and the same values of nMOS size, to avoid redundant computation, CSyn-fp stores only the distinct folding shapes in a lookup table with pMOS and nMOS sizes as key.

2.2.4 Step 2: Search-tree Based Design Space Exploration

Fig. 2.4 shows a conceptual view of our search tree based design space exploration for finding linear orders of minimal length for the transistor pairs in $\mathcal{T}$ of input transistor netlist. Whenever a new node (e.g. (t_1, t_3, t_2) in Fig. 2.4) is expanded in the tree traversal, CSyn-fp extracts a minimal amount of information on the partial order of the least cost (i.e., size) from the information retained in its parent node (e.g., (t_1, t_3)) and will retain the information to be used for its children. (The details will be described in the

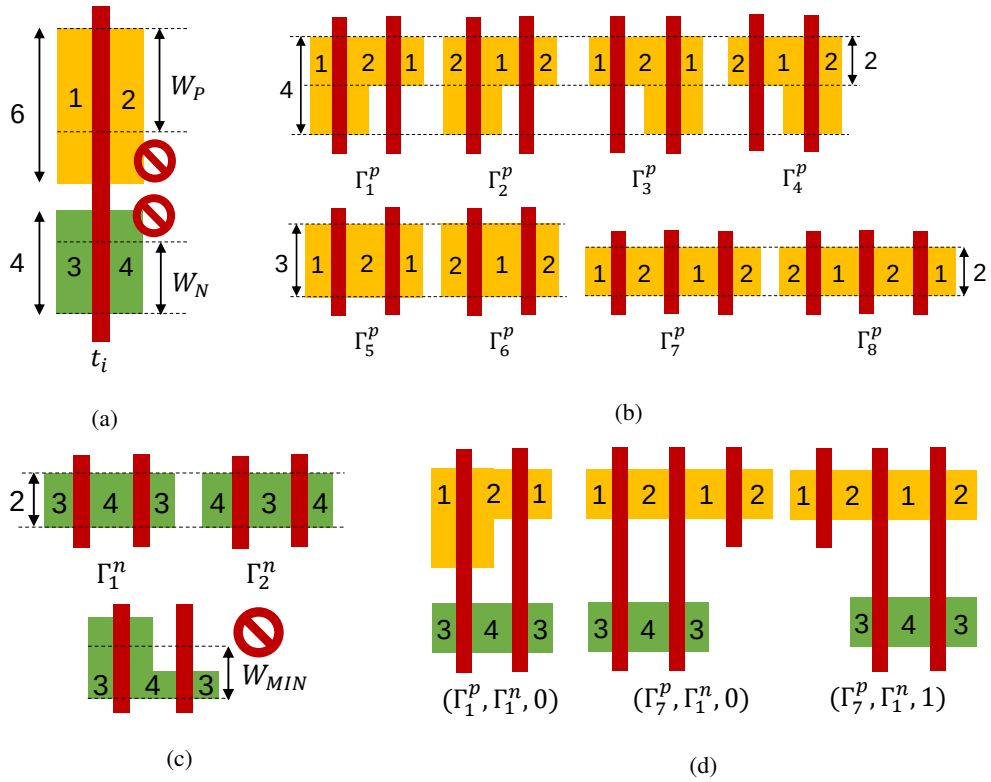


Figure 2.3: Illustration of folding shape generation. (a) Specification of a transistor pair $t_i = (t_i^p, t_i^n) \in \mathcal{T}$ and diffusion width constraints: $size(t_i^p) = 6$, $size(t_i^n) = 4$, $W_P = 4$, $W_N = 3$, and $W_{MIN} = 2$. (b) Generating all folding configurations for t_i^P. (c) Generating all folding configurations for t_i^N. (d) Generating all folding shapes $(\Gamma_{j_1}^p, \Gamma_{j_2}^n, \delta)$.

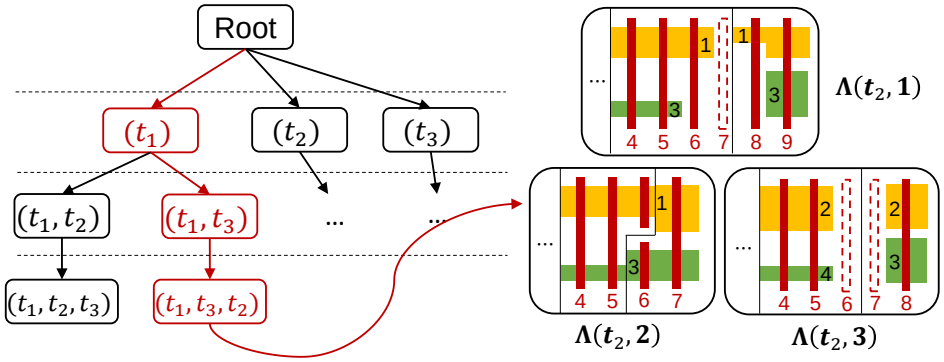


Figure 2.4: A conceptual view of a search tree based design space exploration by CSyn-fp for finding linear orders of minimal length for K transistor pairs. CSyn-fp employs a fast cost computation based on dynamic programming as well as a set of effective speeding up techniques.

following.) In addition, CSyn-fp employs a number of simple but effective speeding up techniques, which will be described in Sec. 2.2.5.

Definition 1 (Cost formulation for partial linear placement of folding shapes)**.** $C[t_{i_1}, t_{i_2}, \cdots, t_{i_{k-1}}, t_{i_k} | t_{i_k} = \Lambda(t_{i_k}, j)]$ *is defined to be the minimal number of poly gates including dummy gates that are required for making the linear order listed as* $t_{i_1}, t_{i_2}, \cdots, t_{i_{k-1}}, t_{i_k}$ *under the condition that folding shape of* t_{i_k} *should be* $\Lambda(t_{i_k}, j)$.

Then, we can derive a recurrence relation for $k \geq 2$:

$$
\begin{aligned}
&C[t_{i_1}, \cdots, t_{i_{k-1}}, t_{i_k} | t_{i_k} = \Lambda(t_{i_k}, j_1)] \\
&= \min_{j_2 \in U} \{ C[t_{i_1}, \cdots, t_{i_{k-1}} | t_{i_{k-1}} = \Lambda(t_{i_{k-1}}, j_2)] + len(\Lambda(t_{i_k}, j_1)) \\
&\qquad + \max(\mu_{DB} \cdot N^{gate}_{dummy},\ \mu_{OD} \cdot L_{OD}) - \alpha \}. \qquad (2.3)
\end{aligned}
$$

where $U = \{1, \cdots, |S_{i_{k-1}}|\}$[3], and μ_{DB} = 0 if the left side of folding shape $\Lambda(t_{i_k}, j_1)$ can abut on the right side of $\Lambda(t_{i_{k-1}}, j_2)$ without a diffusion break and μ_{DB} = 1, otherwise. If the minimum OD jog violation occurs when $\Lambda(t_{i_{k-1}}, j_2)$ and $\Lambda(t_{i_k}, j_1)$

[3] $S_{i_{k-1}}$ is the set of folding shapes for transistor pair $t_{i_{k-1}}$.

abut to each other, μ_{OD} = 1. Otherwise, μ_{OD} = 0. α is the number of vertical line of poly gates saved by abutting the folding shapes.
In addition, for $k = 1$:

$$C[t_i|t_i = \Lambda(t_i, j)] = len(\Lambda(t_i, j)), \forall t_i \in \mathcal{T}. \tag{2.4}$$

Definition 2 (Cost formulation for full linear placement of folding shapes)**.** $C[t_{i_1}, t_{i_2}, \cdots, t_{i_K}]$ *is defined to be* $\min_{j \in U}\{C[t_{i_1}, t_{i_2}, \cdots, t_{i_K} \mid t_{i_K} = \Lambda(t_{i_K}, j)]\}$ *where* $U = \{1, \cdots, |S_{i_K}|\}$.

Then, the minimum among the values of $C[t_{i_1}, t_{i_2}, \cdots, t_{i_K}]$ for every linear order of the K transistor pairs in $\mathcal{T}$ is exactly the quantity of the minimal $Cost$ in *Eq.2.2* in Problem 1.

Fig. 2.5 shows an illustration of calculating the values of $C[\cdots, t_{i_{k-1}}, t_{i_k}|t_{i_k} = \Lambda(t_{i_k}, j_1)]$, j_1 = 1, 2, and 3 in *Eq.2.3* and the chosen (parent) folding shapes by utilizing the parent information of $C[\cdots, t_{i_{k-1}}|t_{i_{k-1}} = \Lambda(t_{i_{k-1}}, j_2)]$, j_2 = 1, 2, and 3. For example, CSyn-fp obtains the value of $C[\cdots, t_{i_{k-1}}, t_{i_k}|t_{i_k} = \Lambda(t_{i_k}, 2)]$, shown on the second row and third column in Fig. 2.5 by computing the sum of the minimum of $C[\cdots, t_{i_{k-1}}|t_{i_{k-1}} = \Lambda(t_{i_{k-1}}, j)] + \max(\mu_{DB_j} \cdot N^{gate}_{dummy}, \mu_{OD_j} \cdot L_{OD})$ - α_j for j = 1, 2, and 3, and $len(\Lambda(t_{i_k}, 2))$, which is $\min\{6 + 0 - 1, 5 + 2 - 0, 6 + 1 - 1\}$ + 2 = 7 where N^{gate}_{dummy} and L_{OD} are set to 2 and 1, respectively, and $len(\Lambda(t_{i_k}, 2)) = 2$ poly gates, as shown in Fig. 2.5. Note that $\alpha_1 = 1$ since abutting $\Lambda(t_{i_k}, 2)$ on $\Lambda(t_{i_{k-1}}, 1)$ saves one vertical space of poly gate. The red arrow in Fig. 2.5 indicates the folding shaping combination with the shortest length.

2.2.5 Speeding up Techniques

• **Pruning partial linear placement of folding shapes**: Let us suppose that the cost computation of the current node in the search tree is completed, producing a set of optimal (conditional) partial linear placement of folding shapes corresponding to $C[\cdots, t_{i_{k-1}}, t_{i_k}|t_{i_k} = \Lambda(t_{i_k}, j)]$, $j = 1, 2, \cdots |S_{i_k}|$.

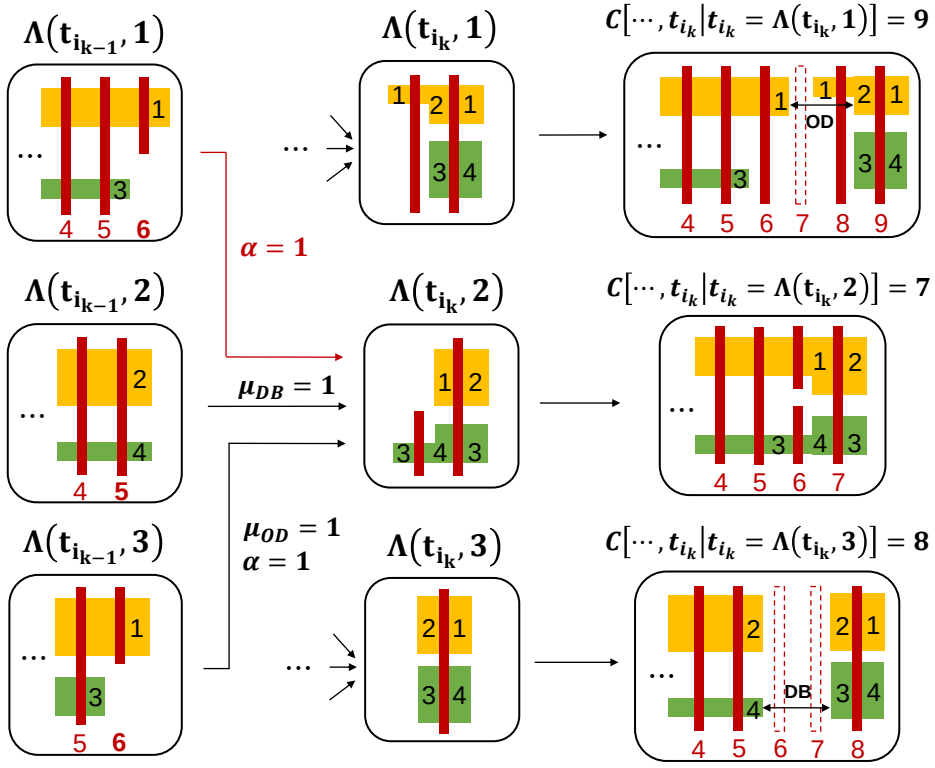


Figure 2.5: Example illustrating the dynamic programming based calculation of the values of $C[\cdots, t_{i_{k-1}}, t_{i_k}|t_{i_k} = \Lambda(t_{i_k}, j_1)]$, j_1 = 1, 2, and 3 together with the corresponding (parent) folding shapes by utilizing the parent information of $C[\cdots, t_{i_{k-1}}|t_{i_{k-1}} = \Lambda(t_{i_{k-1}}, j_2)]$, j_2 = 1, 2, and 3.

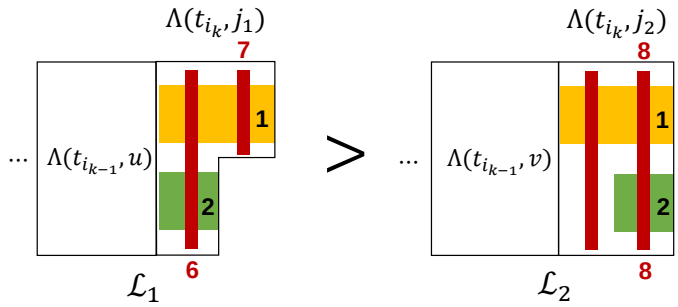


Figure 2.6: Example of showing the relation that $\mathcal{L}_1$ dominates $\mathcal{L}_2$.

Definition 3 (Dominance relation)**.** *For two partial linear placements of folding shapes, $\mathcal{L}_1$ and $\mathcal{L}_2$, corresponding to $C[\cdots, t_{i_{k-1}}, t_{i_k} | t_{i_k} = \Lambda(t_{i_k}, j_1)]$ and $C[\cdots, t_{i_{k-1}}, t_{i_k} | t_{i_k} = \Lambda(t_{i_k}, j_2)]$, it is said that $\mathcal{L}_1$ dominates $\mathcal{L}_2$ if and only if the following four conditions are satisfied. (An illustrated example is shown in Fig. 2.6.)*

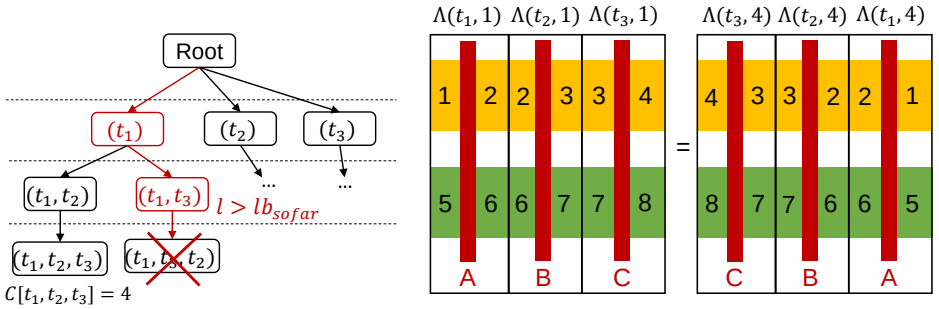
1. $len_p(\mathcal{L}_1) \leq len_p(\mathcal{L}_2)$ where $len_p(\mathcal{L}_i), i = 1, 2$ is the horizontal length, in terms of the number of vertical spaces for poly gates, up to the rightmost pMOS poly gate for $\mathcal{L}_i$.

2. $len_n(\mathcal{L}_1) \leq len_n(\mathcal{L}_2)$ where $len_n(\mathcal{L}_i), i = 1, 2$ is the horizontal length, in terms of the number of vertical spaces for poly gates, up to the rightmost nMOS poly gate for $\mathcal{L}_i$.

3. The two diffusion nets on the right of the rightmost pMOS in $\Lambda(t_{i_k}, j_1)$ and in $\Lambda(t_{i_k}, j_2)$ are the same.

4. The two diffusion nets on the right of the rightmost nMOS in $\Lambda(t_{i_k}, j_1)$ and in $\Lambda(t_{i_k}, j_2)$ are the same.

For example, the partial linear placement corresponding to $C[\cdots, t_{i_k} | t_{i_k} = \Lambda(t_{i_k}, 3)]$ on the bottom row and right column in Fig. 2.5 dominates the order corresponding to $C[\cdots, t_{i_k} | t_{i_k} = \Lambda(t_{i_k}, 1)]$ on the first row and right column in Fig. 2.5. Likewise, as illustrated in Fig. 2.6, the linear order corresponding to the left one dominates the right



(a) Pruning branches based on lower bound.

(b) Avoiding redundant trials of linear placements by cell flipping in placement and folding shape flipping in CSyn-fp.

Figure 2.7: Illustration of search tree pruning based on lower bound and placement symmetry.

one since the conditions are met, as highlighted in Fig. 2.6. Whenever a new node is created in the search tree, CSyn-fp keeps only the most dominating partial linear placements.

• **Pruning based on lower bound of minimal length of linear placement**: From a current node in search tree, CSyn-fp does not expand the search tree further if the following two conditions are satisfied. (An illustrated example is shown in Fig. 2.7(a).)

1. Let $l^p_{current}$ be the smallest horizontal length of pMOS diffusion region among all the partial linear placements corresponding to the current node and $l^p_{unvisit}$ be the total sum of the smallest numbers of poly gates among the pMOS folding configurations of the transistor pairs that are unexplored yet. Then, it should be that $l^p_{current} + l^p_{unvisit} < lb_{sofar}$ where lb_{sofar} is the minimal length among the full linear placements of all transistor pairs explored so far.

2. By the same token, it should be that $l^n_{current} + l^n_{unvisit} < lb_{sofar}$.

• **Eliminating linear placement redundancy by cell flipping**: Since a cell can be flipped in the cell placement and CSyn-fp considers all feasible folding shapes including their flipped ones, a linear placement of all transistor pairs from left to right by

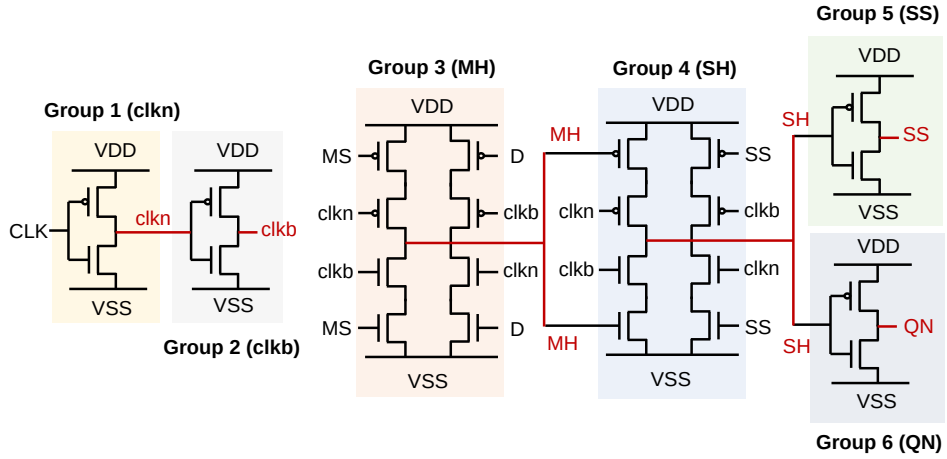


Figure 2.8: Example of *articulation* point based netlist graph partitioning for cell DFFHQNx1 in ASAP7 7nm library.

our CSyn-fp also implicitly includes the consideration of the linear placement from right to left. (An illustrated example is shown in Fig. 2.7(b).) Consequently, it suffices for CSyn-fp to constrain one particular transistor pair in $\mathcal{T}$ to be placed only to the first half of full linear placement.

• **Partitioning netlist for large cells**: For a cell with large number of transistor pairs like flip-flop cells, CSyn-fp partitions the netlist into a number of sub-nets of reasonable size. We observe that some transistor pair in a certain group of transistor pairs in netlist should not necessarily be placed in adjacent to some transistor pair in another group of transistor pairs. For example, the schematic of cell DFFHQNx1 in ASAP 7nm library shown in Fig. 2.8 has 6 such groups. An *articulation* point in netlist graph can be a candidate to split the netlist into multiple parts, each of which can be then treated a single unit for group-level transistor folding and placement.

Precisely, Csyn-fp performs the following steps for netlist partitioning. Firstly, the input netlist is formulated into a graph where each node indicates a distinct net that is connected to at least one of the source or drain of a transistor and each transistor is represented by an edge connecting the nodes corresponding to the nets of source and

drain of the transistor. An articulation point refers to the node where its net connects to both pMOS and nMOS transistors. With each of articulation points as a starting point, Csyn-fp performs the depth first search (DFS) on the graph, and all transistors traversed during the search are partitioned into the same group.

Once the net partitioning is done, CSyn-fp performs a hierarchical two-level transistor folding and placement. At the first level, CSyn-fp finds an optimal solution for each group of transistor pairs. At the second level, CSyn-fp finds an optimal solution for the cell by exhaustively exploring the group placements.

2.2.6 In-cell Routability Estimation

With Step 2, CSyn-fp can find diverse minimum-area transistor placement solutions, but unroutable solutions are useless. We devise an in-cell routability estimation metric which is able to quantify the routing difficulty of the placement solutions from various aspects. The cost function consists of several factors, as follows. (We assume that the placement solution is divided into routing grids, and the pins to be connected by each net are located inside the grid.)

- **HPWL (half parameter wire length)**: For a net, its HPWL is defined to $w + h$ if the (minimal-size) size of bounding box of the net is $w \times h$ routing grids. If HPWL is long, in-cell router requires longer metal wires, which poses a high possibility of inducing dense routing congestion inside the cell. To fit the scale of this value among the cells, the calculated HPWL value is normalized by the HPWL of the cell.

- **Peak horizontal/vertical net density**: For the bounding box of size $w \times h$ routing grids of a net, we define horizontal/vertical/pin density maps as follows. It indicates the expected number of nets passing through the grids for each direction. Horizontal and vertical net densities are $\frac{1}{w}$ and $\frac{1}{h}$, and these values are accumulated to the grids inside the bounding box individually. Since each net

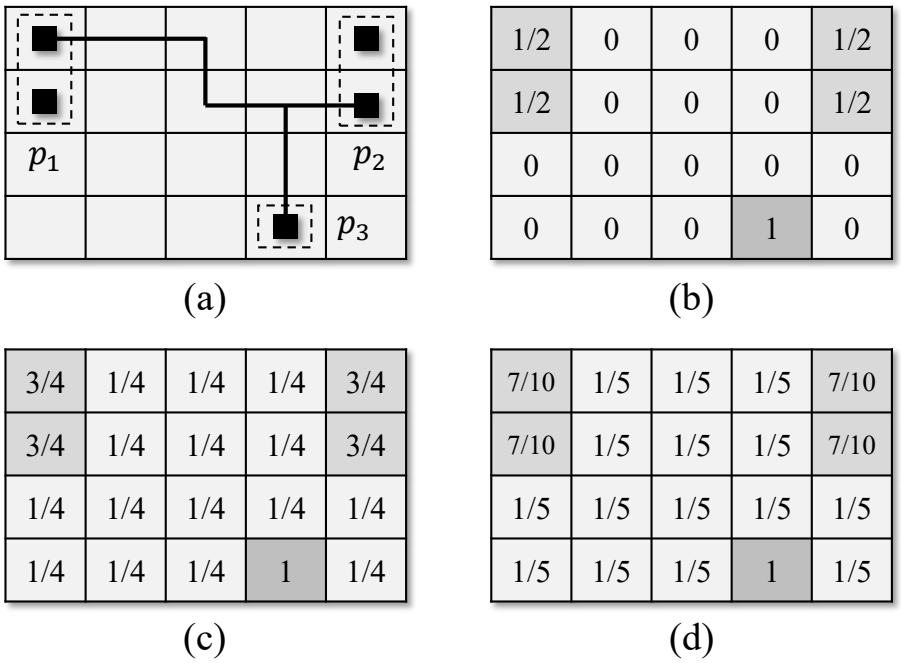


Figure 2.9: Example of calculating horizontal/vertical net density. (a) Multi-pin net. (b) Pin density map. (c) Horizontal net density map. (d) Vertical net density map.

must pass through the grid that contains the pin access point, pin density map calculates the accessing probability of the access points of pin p, which is the $\frac{1}{\#AP(p)}$ where $\#AP(p)$ is the number of access points on pin p. For example, see Fig 2.9 where the size of bounding box of a net is 4×5. Thus, the horizontal and vertical net densities of the net is $\frac{1}{4}$ and $\frac{1}{5}$, respectively. Also, since the net has three pins (p_1, p_2, p_3), and p_1 and p_2 each has two access points, the pin density of p_1 and p_2 each is $\frac{1}{2}$. Pin density is accumulated to the horizontal/vertical net density to consider high congestion around the pins, and if the sum exceeds 1, net density becomes 1 to avoid overestimation. We use the maximum horizontal/vertical net density among the grids as an estimation metric, since in-cell routing failure usually occur in the high congested region.

We formulate the in-cell routability cost as follows:

$$Cost = \alpha \times HPWL + \beta \times M_H + \gamma \times M_V \tag{2.5}$$

where M_H and M_V indicate the maximum horizontal and vertical net densities in the grids, and α, β, and γ are set to 0.5, 0.25, 0.25 in this work. The cost is calculated for all obtained placement solutions, and the placement solutions with low cost become our in-cell routing candidates.

2.2.7 Step 3: In-cell Routing

We implement a grid-based in-cell router using SMT solver inspired by [15]. Our router follows ASAP7 PDK [3] design rules, but any PDK information can be adopted to our router with a slight modification. In-cell routing process comprises the following three steps.

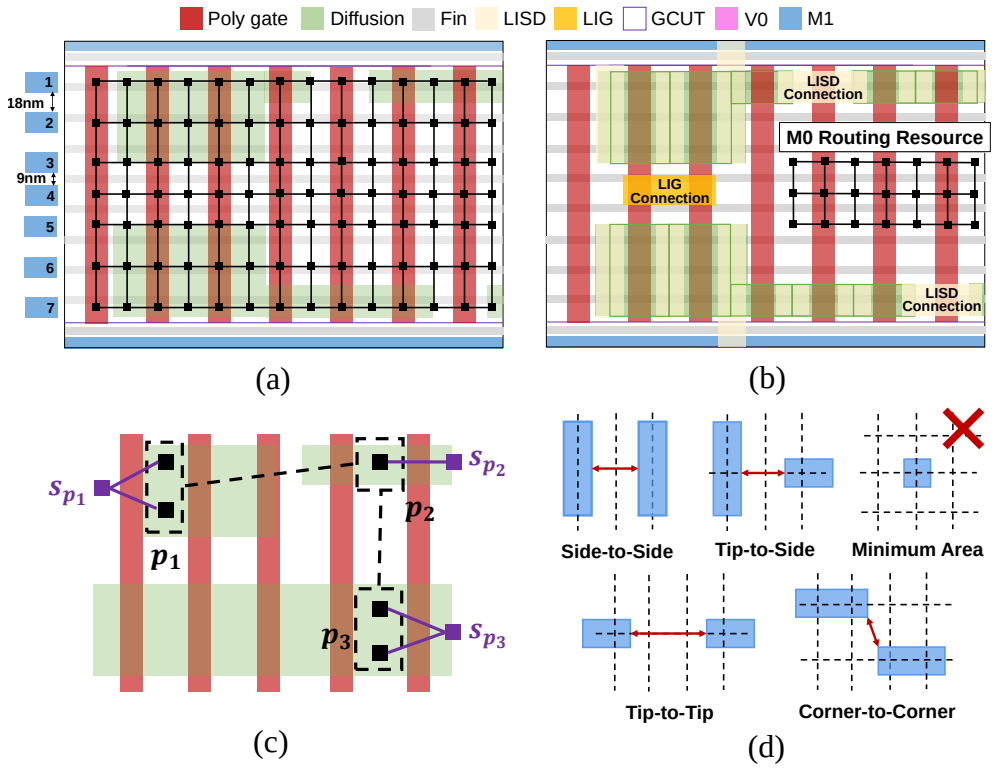


Figure 2.10: Illustration of implementing our SMT-based in-cell router. (a) Grid configuration. (b) Pre-M0 routing. (c) Multi-pin decomposition and supernode for each pin. (d) Design rules in consideration.

Grid configuration

The grid configuration is illustrated in Fig. 2.10(a). We have total of 7 horizontal possible routing tracks: two in pMOS region (tracks 1, 2), two in nMOS region (tracks 6, 7), and three in middle region (tracks 3, 4, 5). The tracks in pMOS and nMOS regions satisfy the side to side spacing rule each other (i.e. 18nm), but the tracks in the middle region are tightly arranged, 9nm spacing between them. There are two reasons: (1) gate poly is usually contacted by its center position. Thus, we have track 4 in the center of the cell and (2) router encourages to maximize the usage of routing resource of the middle region. The vertical track pitch of the grid is half of the gate pitch and the grid line is located at the center of the gate poly and active region.

Pre-MOL connection

There are two MOL (Middle-Of-Line) layers called LISD and LIG in ASAP7 PDK. LIG is used to connect consecutive gate polys with the same signal, and LISD is used to connect distant gates or active regions above the gates. Before using metals for routing, the adjacent active regions and gate polys with the same signal are connected by LISD and LIG, respectively like that in Fig 2.10(b). Then, we identify MOL routing resource that does not cause design rule violation with other MOL metals. This region also be considered as possible MOL routing region in the next metal routing step.

Metal routing by SMT

Assume that a net has a set of pins $P(n) = \{p_1, p_2, ..., p_k\}$ to be connected. Each pin p_i may have multiple access points, and when routing, it is necessary to be connected to at least one of these points. To ensure this, for all pins in $P(n)$, we create a supernode s_{p_i} which is connected to all access points of pin p_i, and SMT formulation ensures that these supernodes should be connected. Fig. 2.10(c) shows an example of the supernode generation. Multi-pin nets are decomposed into multiple two-pin nets by Kruskal algorithm [60] to minimize the wirelength. Also, we consider various de-

Table 2.1: Notations for the SMT-based in-cell router of the CSyn-fp framework.

Notation	Description
$G(V, E)$	Grid graph
$e(v, v')$	Edge between node v and v'
$N(v)$	A set of neighboring nodes of node v
$P(p)$	A set of possible pin location points of pin p
$PL(v, p)$	0/1 variable if pin p is located on node v
$h(x, y, z, n)$	0/1 variable if horizontal edge between (x, y, z) and $(x + 1, y, z)$ is connected by net n
$v(x, y, z, n)$	0/1 variable if vertical edge between (x, y, z) and $(x, y + 1, z)$ is connected by net n
$i(x, y, z, n)$	0/1 variable if via edge between (x, y, z) and $(x, y, z + 1)$ is connected by net n
$m(x, y, z, n)$	0/1 variable if node (x, y, z) is possessed by net n

sign rules like side-to-side (S2S) spacing, tip-to-side (T2S) spacing, minimum area rule (MAR), tip-to-tip (T2T) spacing, corner-to-corner (C2C) spacing, and via spacing rule. Detailed illustration of these rules are shown in Fig. 2.10(d). Our SMT formulation for ensuring net connection and considering the design rules are identical to that in [15], and we set the objective function of SMT to minimizing metal 2 usage in the first priority and metal 1 wirelength usage in the second priority. We assume that MOL and metal 1 layer use 2D routing and metal 2 layer routing is performed by 1D horizontal routing.

Detailed SMT formulation of our in-cell router is as follows. Basic notations are defined in Table 2.1.

• **Vertex exclusiveness**: Definition of metal grid is shown in Eq. 2.6.

$$m(x, y, z, n) = \bigvee_{e \in E(x,y,z,n)} e, \quad \forall (x, y, z) \tag{2.6}$$

where $E(x, y, z, n) = \{h(x - 1, y, z, n), h(x, y, z, n), v(x, y - 1, z, n), v(x, y, z, n), i(x, y, z - 1, n), i(x, y, z, n)\}$. Two different net can not possess a node $v \in G(V, E)$ simultaneously.

$$\sum_{\forall n} m(x, y, z, n) \leq 1, \quad \forall (x, y, z) \tag{2.7}$$

• **Ensuring connectivity**: For mth two-pin net of net n, source supernode s_n^m and destination supernode d_n^m should have one active adjacent edge.

$$\sum_{v \in N(s_n^m)} e_n^m(s_n^m, v) = 1, \sum_{v \in N(d_n^m)} e_n^m(d_n^m, v) = 1 \tag{2.8}$$

and grid nodes have 0 or 2 active edges.

$$\sum_{e \in A^m(x,y,z,n)} e = 0 \; or \; 2, \quad \forall (x, y, z) \tag{2.9}$$

where $A^m(x, y, z, n) = E^m(x, y, z, n) \cup \{e^m(v, s_n^m) | v \in V(s_n^m)\} \cup \{e^m(v, d_n^m) | v \in V(d_n^m)\}$. Routing result of net n can be obtained as in Eq. 2.10.

$$\bigvee_{\forall e=(v,v') \in E} e_n^m(v, v') = e_n(v, v') \tag{2.10}$$

• **Pin allocation**: For I/O pin p, location of the pin must be guaranteed.

$$\sum_{v \in P(p)} PL(v, p) = 1, (PL(v, p) == 1) \implies (m(v, p) == 1) \tag{2.11}$$

• **Side-to-side spacing**: Vertical metal cannot be placed adjacently.

$$S2S = \neg(v(x, y, z) \wedge v(x + 1, y, z)), \quad \forall (x, y, z) \tag{2.12}$$

• **Tip-to-side spacing**: Horizontal metal tip and vertical metal should maintain enough space between them.

$$\begin{aligned} T2S_1 &= v(x, y, z) \wedge v(x, y + 1, z) \wedge \neg h(x, y + 1, z) \wedge h(x + 1, y + 1, z) \\ T2S_2 &= v(x + 2, y, z) \wedge v(x + 2, y + 1, z) \wedge \neg h(x + 1, y + 1, z) \wedge h(x, y + 1, z) \\ T2S &= \neg(T2S_1 \vee T2S_2), \quad \forall (x, y, z) \end{aligned} \tag{2.13}$$

• **Tip-to-tip spacing**: Two Horizontal metal tip should be placed two edges apart.

$$T2T = \neg(h(x, y, z) \wedge \neg h(x + 1, y, z) \wedge h(x + 2, y, z)), \quad \forall (x, y, z) \tag{2.14}$$

• **Corner-to-corner spacing**: Two near vertices of the two horizontal metals must be far apart.

$$\begin{aligned} C2C_1 &= h(x,y,z) \wedge h(x+2,y+1,z) \wedge \neg h(x+1,y,z) \wedge \neg h(x+1,y+1,z) \\ &\quad \wedge \neg v(x+1,y,z) \wedge \neg v(x+2,y,z) \\ C2C_2 &= h(x+2,y,z) \wedge h(x,y+1,z) \wedge \neg h(x+1,y,z) \wedge \neg h(x+1,y+1,z) \\ &\quad \wedge \neg v(x+1,y,z) \wedge \neg v(x+2,y,z) \\ C2C &= \neg(C2C_1 \vee C2C_2), \quad \forall(x,y,z) \end{aligned} \tag{2.15}$$

• **Minimum area rule**: The area of the metal should not be too small.

$$\begin{aligned} MAR = m(x,y,z) &\wedge \neg h(x-1,y,z) \wedge \neg h(x,y,z) \wedge \neg v(x,y-1,z) \\ &\wedge \neg v(x,y-1,z), \quad \forall(x,y,z) \end{aligned} \tag{2.16}$$

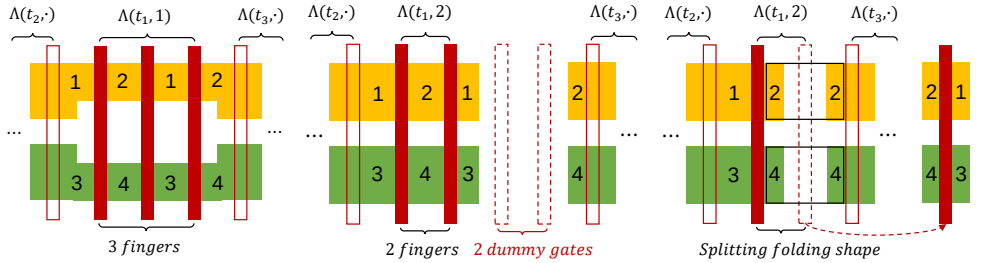
To ensuring design rule violation during in-cell routing, constraint of Eq. 2.17 is included in SMT formulation.

$$\neg S2S \wedge \neg T2S \wedge \neg T2T \wedge \neg C2C \wedge \neg MAR \tag{2.17}$$

It should be noted that other than the proposed SMT-based router, we can apply any of conventional in-cell routers to the solutions of transistor folding and placement produced by Steps 1 and 2 in CSyn-fp.

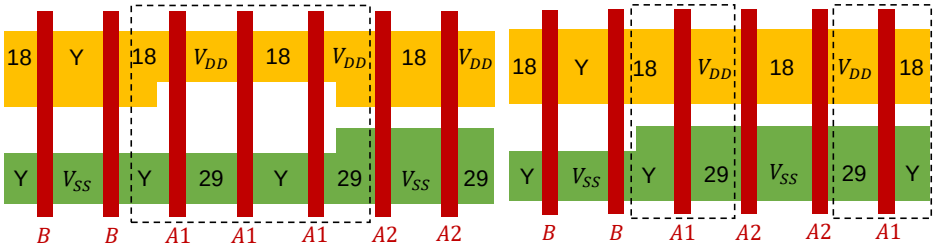
2.2.8 Step 4: Splitting Folding Shapes

CSyn-fp has used the folding shapes as basic units since treating the fingers in the folding shapes individually requires additional metal resource to connect the fingers apart. However, splitting the fingers in folding shapes may reduce the cell size, as illustrated in Fig. 2.11. Fig. 2.11(a) shows a section of transistor folding and placement, in which folding shape $\Lambda(t_1, 1)$ consists of three fingers. This is because if transistor pair t_1 were replaced with $\Lambda(t_1, 2)$ of two fingers, as shown in Fig. 2.11(b), two dummy poly gates ($N^{gate}_{dummy} = 2$) should be needed to break diffusion. However, if



(a) (b) (c)

Figure 2.11: Impact of splitting folding shape on cell size. (a) t_1 is placed in three fingers. (b) If t_1 is folded by two fingers, the diffusion break between t_1 and t_3 is inevitable. (c) Diffusion break can be resolved by placing two fingers of t_1 separately.



(a) Layout before folding shape splitting (b) Layout after folding shape splitting

Figure 2.12: Layouts of AOI21x1 in ASAP 7nm library [3] produced before and after the application of our refinement step in CSyn-fp.

one finger in $\Lambda(t_1, 2)$ (generally half of the fingers in the folding shape of even number of fingers) is displaced as shown in Fig. 2.11(c), the dummy gates are not needed, thereby providing a potential saving of cell area.

CSyn-fp implements this idea as an optional step as follows. For each folding shape with an odd number of fingers in a layout solution, $\mathcal{L}_i$, obtained in Step 3 of CSyn-fp, we update the folding shape to have one fewer number of fingers and then perform an iterative process of swapping fingers in a short distance while controlling the number of iterations. We repeat this process for all folding shapes of odd number of fingers in $\mathcal{L}_i$. For example, Figs. 2.12(a) and (b) show the layouts of cell AOI21x1 in ASAP 7nm library [3] produced before and after the application of our refinement step

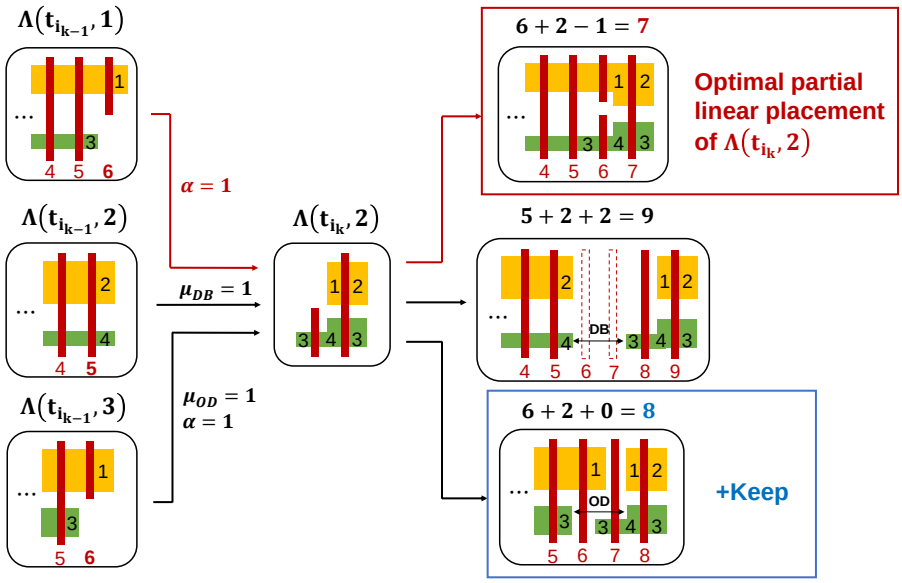


Figure 2.13: Example finding optimal partial placements of $\Lambda(t_{i_k}, 2)$ when the area relaxation constraint k is s t to 1. The node keeps not only the 7 CPP placements but also the 8 CPP placement solutions.

(i.e., folding shape splitting) in CSyn-fp, respectively. It is shown that by splitting the folding shape with dashed box and swapping one split finger labeled $A1$ with fingers labeled $A2$, CSyn-fp reduces the total number of poly gates from 7 to 6.

2.2.9 Step 5: Relaxing Minimum-area Constraints

The minimum area solutions can be found through Steps 1 and 2, but all of these placements may be unroutable or using too much metal 2 resources. This situation occurs a lot in sequential cells where there are many nets to be connected and exist crossover signals. This step tries to find placement solutions with a little larger area than minimum-area solution but better in-cell routability by relaxing the minimum area related constraint.

We define k to the area relaxation constraint such that we want to find the place-

ment solutions with k more gate polys than that of the minimum-area solution. To explore the solutions, we simply keep the partial placements that have k more polys than minimum in the dynamic programming calculation process for each node during the search tree. For example in Fig. 2.13, optimal partial placement of $\Lambda(t_{i_k}, 2)$ is the right top partial placement with 7 polys, but when the area constraint k is relaxed to 1, the right bottom placement with 8 polys can generate the placement solution that is 1 CPP more than the minimum one. When the expansion on the tree search reaches leaf nodes, Csyn-fp collects the complete placement solutions such that their widths are smaller than or equal to $w_{min} + k$ where w_{min} is the minimum width obtained by the first run of our algorithm.

Since this step forces the nodes in the search tree to keep partial placements more than that when finding minimum-area placements only, additional runtime overhead during the dynamic programming based cost calculation inside the nodes is necessary. To speedup the second run of our algorithm, we fix the lower bound of the search tree as $w_{min} + k$. This procedure is able to significantly reduce the overall runtime through efficient pruning of the search space that is unlikely to contain placement solutions we look for.

2.3 Experimental Results

We implemented CSyn-fp using C++ on a linux machine with Intel i7-8700K 4.7GHz CPU and 64GB memory. CSyn-fp of transistor folding and placement is applied to the ASAP 7nm standard cell library publicly available in [1] in which the design rule parameters have been set as: $N^{gate}_{dummy} = 2$, $W_{MIN} = 1$, $W_P = 3$, $W_N = 3$ and $L_{OD} = 1$. CSyn-fp synthesizes 172 cells in the ASAP 7nm library, and in-cell router is implemented with Z3 SMT solver [61].

Table 2.2: Comparison of 172 all cells in ASAP 7nm cell library [1] with the cells produced by CSyn-fp in terms of the number of CPPs (contacted poly pitches) (i.e., cell width) and the number of cells with in-cell routing completion together with the portion of M2 wire usage.

Cells		ASAP [1]				ASAP's placement + Csyn-fp's routing				Csyn-fp				
Type	#Cells	Size	Routing completion			Size	Routing completion			Size		Routing completion		
		#CPPs	M1 only	M1+M2	M2 wire	#CPPs	M1 only	M1+M2	M2 wire	#CPPs	Impr.	M1 only	M1+M2	M2 wire
AO/AOI	42	9.52	42	0	-	9.52	40	2	4.2%	9.19	**3.50%**	41	1	3.7%
OA/OAI	34	9.26	34	0	-	9.26	32	2	9.0%	9.03	**1.27%**	33	1	3.7%
AND/NAND	22	9.05	22	0	-	9.05	21	1	1.8%	8.95	**1.00%**	21	1	2.6%
OR/NOR	22	8.43	22	0	-	8.43	21	1	10.5%	8.43	-	22	0	-
XOR/XNOR	6	10.7	4	2	19.5%	10.7	2	4	19.4%	10.7	-	4	2	6.7%
BUF/INV	28	9.46	28	0	-	9.46	28	0	-	9.46	-	28	0	-
HA	5	6.20	5	0	-	6.20	4	1	8.2%	6.20	-	5	0	-
FA	1	14.0	0	1	41.0%	14.0	0	1	41.0%	14.0	-	0	1	35.3%
Latch	6	16.0	0	6	19.8%	16.0	0	6	3.0%	15.0	**6.25%**	0	6	15.5%
DFF	8	22.0	0	8	15.6%	22.0	0	8	14.4%	19.5	**11.4%**	0	8	21.7%

2.3.1 Comparison with ASAP 7nm Cell Layouts

Table 2.2 shows a comparison of 172 all cells in ASAP 7nm cell library [1] with the cells produced by CSyn-fp. Since CSyn-fp aims not only to find a minimal-area transistor placement solution but also to minimize the usage of metal 2 in in-cell routing, to validate the efficacy of CSyn-fp, we also applied Csyn-fp's in-cell router to the placement in ASAP cells. In Table 2.2, the cell size is specified in terms of the number of CPPs (contacted poly pitches) on the cell; the numbers of cells successfully routed by using metal 1 layer only and by using both of metal 1 and metal 2 layers are specified in the *M1 only* and *M1+M2* columns, respectively; the portion of wire usage on the metal 2 layer for in-cell routing is shown in the *M2 wire* column.

For combinational cells, CSyn-fp reduces area on AO/AOI, OA/OAI, and AND/NAND types at the expense of using wire on metal 2 layer. In particular, among six ASAP cells of XOR/XNOR type, two ASAP cells use metal 2 wire while Csyn-fp generates in-cell routing on the six cells of the same transistor placement as that of ASAP cells, in which Csyn-fp uses metal 2 wire on four cells. This is mainly because of the adoption of grid-based routing structure by Csyn-fp. Overall, Csyn-fp is able to produce placement outcomes with better in-cell routability by utilizing our routability estimation metric. On the other hand, by applying Csyn-fp to both of transistor placement and in-cell routing on the six XOR/XNOR types it produces layouts among which only two types use metal 2 wire, which implies that the placements produced by Csyn-fp exhibit better in-cell routability than that of the ASAP cells. Particularly, for XOR/XNOR and FA cells, CSyn-fp produces layouts with less usage of metal 2 wire over that of the corresponding ASAP ones.

For sequential cells, Csyn-fp reduces cell area by 6.25% for latches and by 11.4% for DFFs on average. For latch cells, Csyn-fp improves both of cell area ($16.0 \rightarrow 15.0$) and metal 2 usage ($19.8\% \rightarrow 15.5\%$). Figs. 2.14(a) and (b) compares the layouts of the latch DHLx3 in ASAP 7nm library and produced by Csyn-fp. CSyn-fp is able to produce layout with 15 CPPs, which is 11.8% ($17 \rightarrow 15$) smaller than that of ASAP

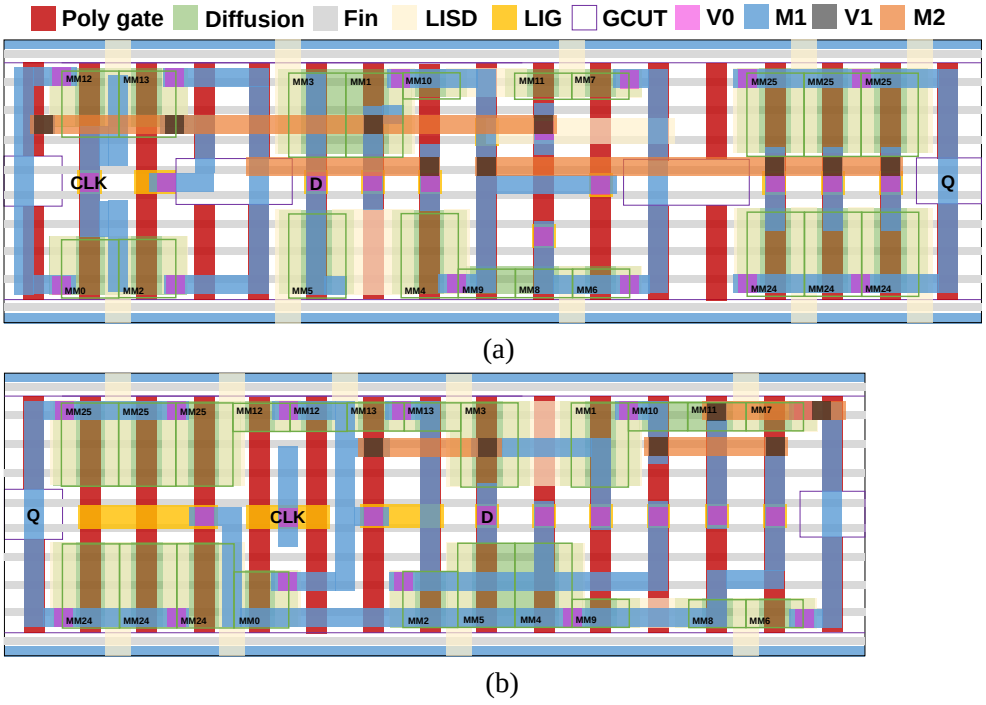


Figure 2.14: Layout comparison for latch DHLx3 in [1]. CSyn-fp synthesizes cells with 11.8% smaller area with less metal 2 wire usage. (a) Layout in ASAP cell library. (b) Layout by CSyn-fp.

Table 2.3: Running time of CSyn-fp with the option of static or dynamic folding for netlists of the cells in ASAP 7nm library.

Cells	#FETs	CSyn-fp with static folding		CSyn-fp with dynamic folding	
		#CPPs	Time (s)	#CPPs	Time (s)
AND4x2	10	16	1.497	15	0.586
AO22x1	10	9	0.038	8	0.128
AO322x2	16	13	0.334	12	0.374
AOI211x1	8	12	0.477	11	0.408
AOI221x1	10	14	2.801	13	3.049
AOI222xp33	12	10	0.183	9	0.149
OA31x2	10	14	0.756	13	2.576
OAI221xp5	10	9	0.015	8	0.029
Others (164 Cells)	2∼32	10.6	0.720	10.6	3.314

cell while using less wire usage on metal 2 layer. For DFF cells, Csyn-fp reduces area by 11.4%, but it uses 6.1% more metal 2 wire usage.

2.3.2 Effectiveness of Dynamic Folding

To activate dynamic folding, CSyn-fp generates various folding shapes for each transistor pair in Step 1. These folding shapes enable to find a minimum area transistor placement solution through the exploration of folding options. To assess the effect of dynamic folding on area improvement, we compare the area of cells produced by applying static folding with that produced by dynamic folding. For static folding, we generate only the folding shapes of minimum number of fingers on each of pMOS and nMOS transistors for every transistor pair in Step 1 of our algorithm.

Table 2.3 shows the running time of CSyn-fp when cell layouts are synthesized with the option of static folding and dynamic folding. Total of 8 cells show area improvement when dynamic folding is performed, and the rest have the same area of cell layout with static folding option. Since dynamic folding explores more folding

configurations than static folding, the runtime on transistor placement is longer when performing dynamic folding. On average, Csyn-fp takes about 0.7 seconds using static folding and about 3.2 seconds using dynamic folding. In some cells such as AND4x2, AOI211x1 and AOI222xp33, using static folding takes longer time than using dynamic folding, which is caused by a quick finding on the optimal lower bound in the search tree exploration.

Figs. 2.15(a) and (b) show the layouts of cell AOI211x1 produced by using static and dynamic folding, respectively. It shows that using static folding option uses two fingers for every transistor of size 6, causing a diffusion break in the middle of the layout, resulting in generating cell with 12 CPPs. On the other hand, by using dynamic folding, transistor MM6 in the black box in Figs. 2.15(b) is folded into three fingers to avoid diffusion break, resulting in generating cell with 11 CPPs. Note that since the design space for dynamic folding properly covers the design space of static folding, it is required to use dynamic folding if a minimal-area cell is the primary objective.

2.3.3 Effectiveness of Speeding Up Techniques

Table 2.4 shows the running time comparison of the combinational cells when the three speeding up techniques in Sec. 2.2.5 are incrementally applied. When the dominance elimination technique is applied, the runtimes are boosted by 1.9x for all test cases on average. As explained in Sec. 2.2.5, CSyn-fp efficiently removes redundant partial placements on the internal nodes of search tree, and this technique shows a great efficiency for the cells which have transistor pairs with numerous folding shapes. The significant runtime improvement on AND/NAND types is mainly due to the cell "NAND3x2" whose transistor pairs have 18 fins on pMOS and 6 fins on nMOS. Thus, each pair has a large number of folding shapes, showing 98.6x runtime improvement on the cell by this technique. Since the eliminating symmetry technique reduces the size of the entire search space by half, by applying it combined with the redundant partial placement removal technique, the runtime is improved by 3.3x on average. In

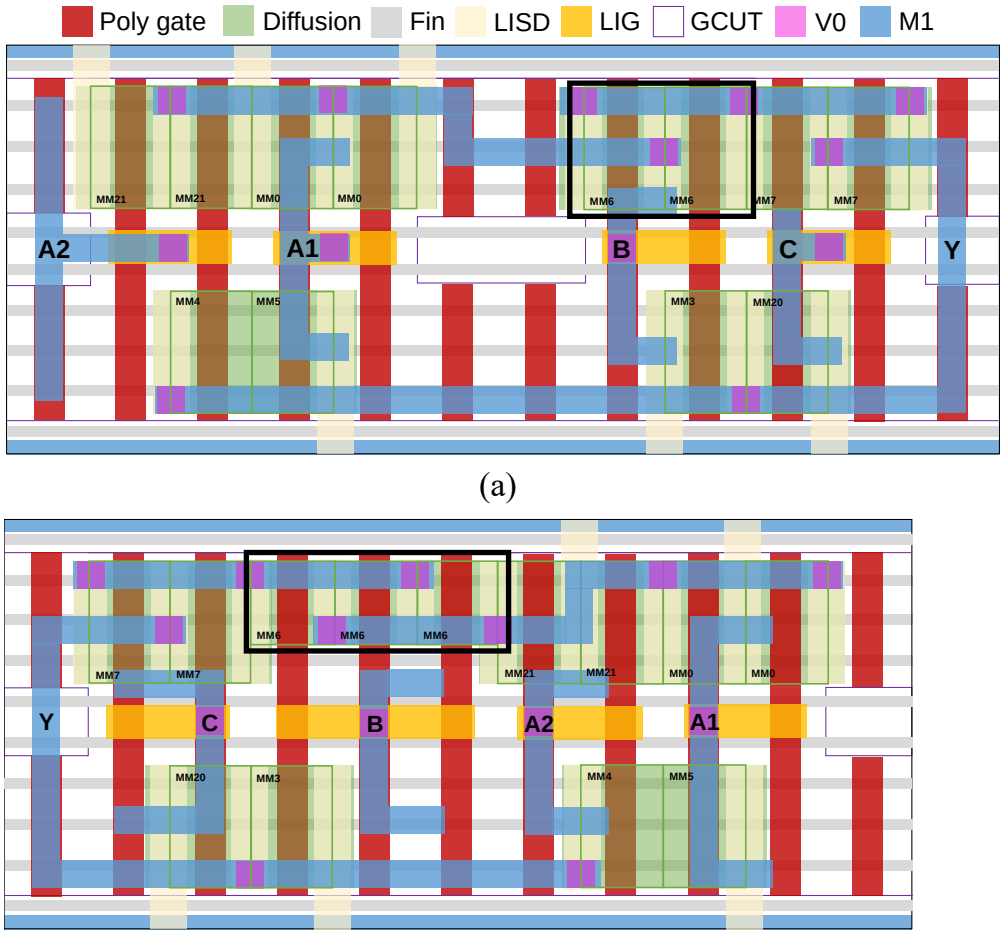


Figure 2.15: Layout comparison for cell AOI211x1 between static folding and dynamic folding options. (a) Layout with 12 CPPs produced by using static folding. (b) Layout with 11 CPPs produced by using dynamic folding.

Table 2.4: Running time comparison of combinational cells when gradually applying the speeding up techniques (SYM: Eliminating symmetry, DOM: Eliminating dominant partial placement, and BOUND: Search tree bounding based on lower bound).

Cells		#FETs		None	DOM		SYM+DOM		SYM+DOM+BOUND	
Type	#Cells	Min.	Max.	Time (s)	Time (s)	Speedup	Time (s)	Speedup	Time (s)	Speedup
AO/AOI	42	8	20	34.542	12.086	2.86×	11.330	3.05×	9.379	3.68×
OA/OAI	34	8	20	20.319	16.934	1.20×	8.753	2.32×	3.196	6.36×
AND/NAND	22	4	12	19.815	0.323	61.35×	0.219	90.48×	0.208	95.26×
OR/NOR	22	4	12	0.175	0.099	1.77×	0.094	1.86×	0.046	3.80×
XOR/XNOR	6	10	12	7.964	6.842	1.16×	3.256	2.45×	1.460	5.45×
BUF/INV	28	2	12	0.047	0.036	1.31×	0.025	1.88×	0.019	2.47×
HA/FA	6	10	24	0.589	0.33	1.78×	0.217	2.71×	0.084	7.01×
Avg.						**1.90**×		**3.29**×		**5.03**×

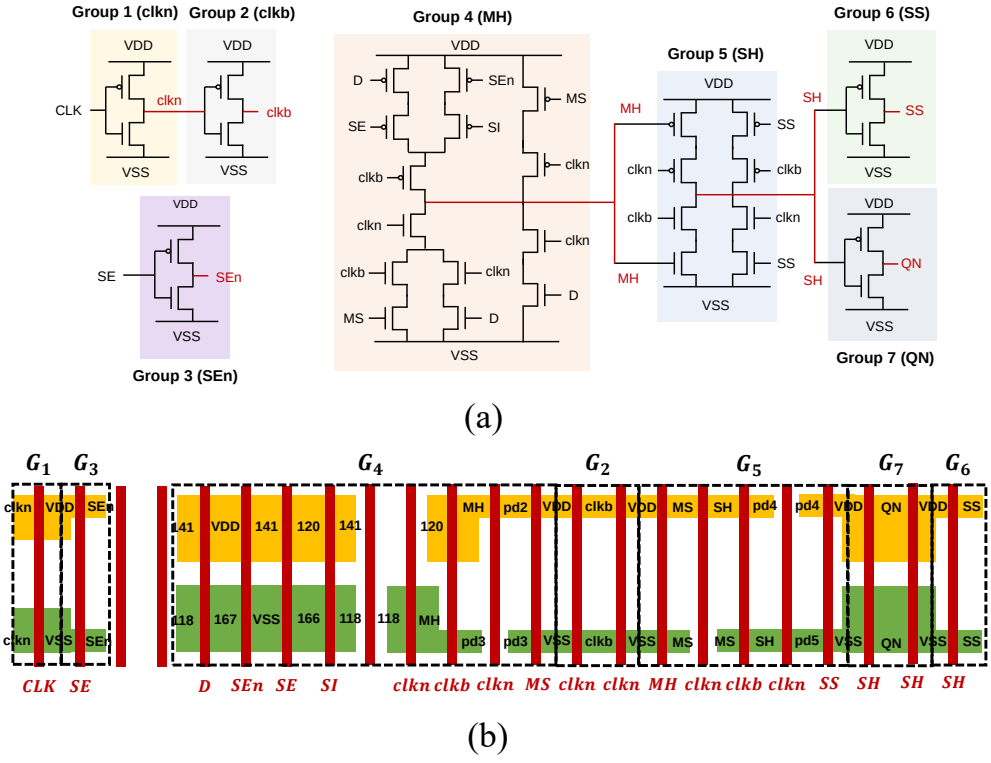


Figure 2.16: Result of netlist partitioning for cell SDFHx2 in ASAP 7nm cell library. (a) CSyn-fp divides the entire netlist into seven groups. (b) The minimum area placement solution found by two-level placement (25 CPPs).

addition, the search tree bounding technique efficiently reduces meaningless search to the nodes that are very unlikely to produce optimal solutions. Thus, by applying all three techniques, the runtime is boosted by 5.0x on average.

Table 2.5 shows the running time spent by CSyn-fp with the option of no-partitioning or partitioning based on articulation points for netlists of 15 sequential cells in ASAP 7nm library. These cells are D flip-flop and scanned D flip-flop, which are the largest sequential cells in the library. The comparison indicates that CSyn-fp employing articulation point based netlist partitioning can generate layouts for large sequential cells with significant runtime improvement. Furthermore, the layout optimality in terms of

Table 2.5: Running time of CSyn-fp with the option of no-partitioning or partitioning by articulation points for netlists of sequential cells in ASAP 7nm library.

Cells	#FETs	CSyn-fp without partitioning		CSyn-fp with partitioning	
		#CPPs	Time (s)	#CPPs	Time (s)
DFFHQNx1	24	18	7.903	18	0.295
DFFHQNx2	24	18	21.41	18	0.837
DFFHQNx3	24	20	7.674	20	0.272
DFFHQNx4	24	22	447.2	22	3.899
DFFLQNx1	24	18	8.476	18	0.266
DFFLQNx2	24	18	17.85	18	0.717
DFFLQNx3	24	20	7.091	20	0.252
DFFLQNx4	24	22	501.1	22	3.937
SDFHx1	32	25	30798.9	25	3.792
SDFHx2	32	25	17665.0	25	1.096
SDFHx3	32	27	11935.7	27	4.767
SDFHx4	32	27	14094.7	27	1.145
SDFLx1	32	25	15205.2	25	3.525
SDFLx2	32	25	12325.3	25	0.998
SDFLx3	32	27	14317.8	27	4.438
SDFLx4	32	27	13687.5	27	1.149
Avg. Speedup					**4176**×

#CPPs still holds for all netlists. Fig 2.16 shows a partitioning example of SDFHx2 in the library and the placement result produced by Csyn-fp through two-level hierarchical placement. Csyn-fp divides the netlist into seven different groups automatically, thereby producing a minimum-area placement solution in about 1 second.

Since our search tree based algorithm explores all possible permutations of transistor pairs, the runtime complexity of the search without net partitioning is $O(N!)$ where N is the number of transistor pairs. On the other hand, the search space of two-stage placement after partitioning is bounded by $O(N_g! \cdot G!)$ where G is the number of partitioned groups and N_g indicates the maximum number of transistor pairs among the groups. Therefore, if the partitioned netlist has a larger number of groups and the transistor pairs are evenly distributed among the groups, a significant runtime reduction can be expected. Note that the netlist partitioning technique shows an excellent efficiency on the sequential cells since they have multiple transmission gates. Considering that articulation points can be found on the source/drain of these gates, the netlist of sequential cells were partitioned into a large number of groups (i.e. more than 5 groups in our experiment), thereby greatly reducing runtime. On the other hand, all the combinational cells are partitioned into up to 2 groups, and for each of most cells, one of the groups is just an inverter with one transistor pair, revealing no gain on efficiency on runtime.

2.3.4 Impact of Splitting Folding Shape

Table 2.6 shows the running time of Csyn-fp according to the choice of performing splitting folding shapes for all cells in the ASAP 7nm library. As explained in Sec. 2.2.8, our folding shape generation assumes that all folded transistors of the same transistor pair should be placed adjacently. Therefore, it is not possible to find the case where the diffusion break can be eliminated when the folding shape is split and placed separately. Experimental results show that we can find the minimum area cell layout with folding shape splitting for the 10 cells in Table 2.6, and its runtime overhead is

Table 2.6: Running time of CSyn-fp according to the choice of performing splitting folding shape for netlists of the cells in ASAP 7nm library.

Cells	#FETs	CSyn-fp without splitting		CSyn-fp with splitting	
		#CPPs	Time (s)	#CPPs	Time (s)
AND2x4	6	11	0.028	10	0.087
AND2x6	6	13	0.025	12	0.067
AO31x2	10	14	0.171	12	2.180
AOI21x1	6	9	0.003	8	0.135
AOI22x1	8	12	0.016	10	0.705
AOI31xp67	8	11	0.024	10	0.426
OAI21x1	6	9	0.005	8	0.130
OAI22x1	8	12	0.023	10	0.804
OAI31xp67	8	11	0.069	10	0.929
OR2x6	6	13	0.006	12	0.188
Others (162 Cells)	2∼32	10.6	1.089	10.6	3.214

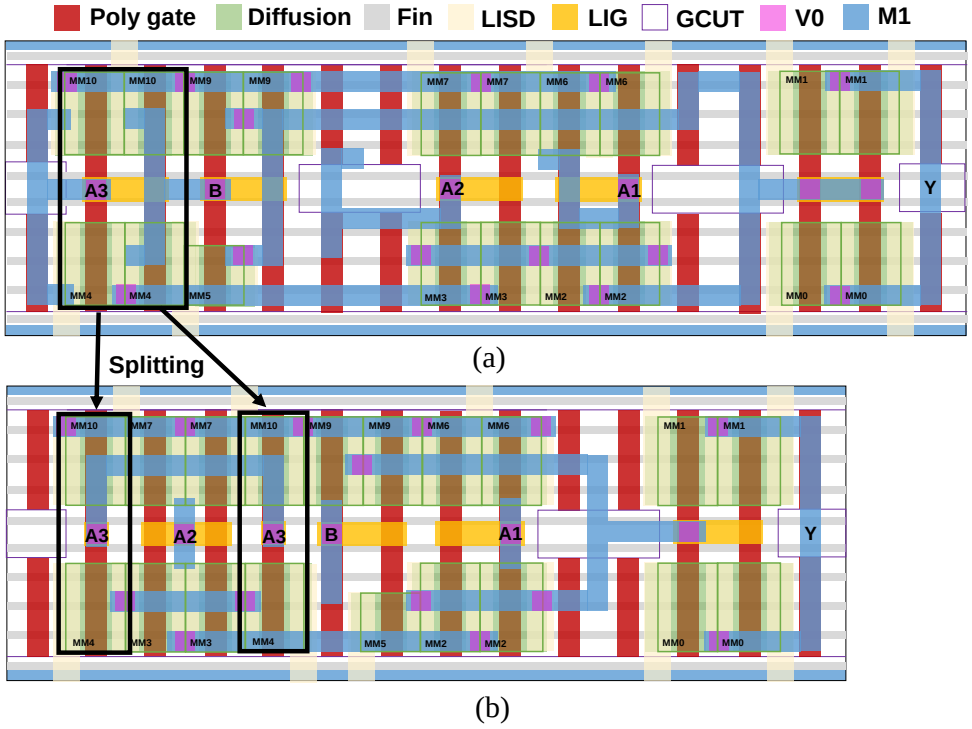


Figure 2.17: Cell layout of AO31x2 in ASAP 7nm library produced by Csyn-fp. (a) Not performing folding shape splitting. (b) Performing folding shape splitting.

not significant because the runtime for the most cells are within 1 second. For the other cells, splitting folding shape stage does not bring area improvement, and the runtime overhead is about 3x over that without performing this step.

This process is especially effective for combinational cells, in which transistor pairs have a multiple number of fingers. Figs 2.17(a) and (b) show layouts of AOI31x2 according to the choice of performing folding shape splitting. If transistor pairs with A3 gate signal were split, Csyn-fp is able to save 2 CPP ($16 \rightarrow 14$) by eliminating one diffusion break in Fig 2.17(a). However, these pairs cannot be routed through MOL layers due to A2 I/O pin between them, which may exacerbate in-cell routability since these need to be connected using upper metal layers.

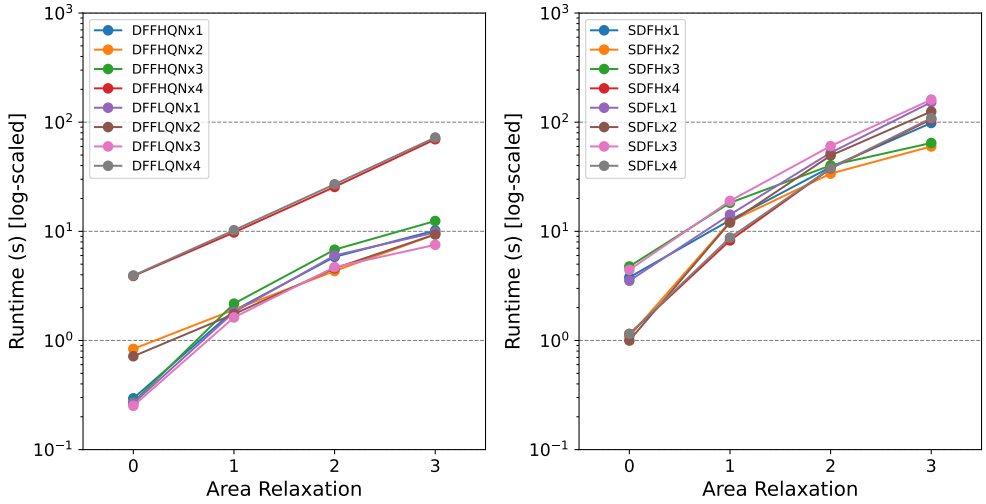


Figure 2.18: Trend of runtime increment according to area relaxation constraint for DFF and SDFF cells. If area relaxation is k, Csyn-fp searches transistor placement solutions with up to k more CPPs than the CPPs of minimum-area solution.

2.3.5 Runtime Analysis According to Area Relaxation

If a minimum-area transistor placement is unroutable or using too much metal 2 resources, we should find a placement solution with good routability at the cost of increasing cell size. One valuable capability of Csyn-fp is that it can search placement solution of various cell widths by performing Step 5, but it requires additional runtime. Fig 2.18 shows the runtime increment trend according to the area relaxation constraint for D flip-flop and scanned D flip-flop cells. If the area relaxation is k, Csyn-fp searches placement solution with up to k more CPPs than the CPPs of minimum-area solution. As the area relaxation constraint value increases, the total runtime increases exponentially. However, if the area relaxation is 2 or less, the runtime is within 1 minute. Our experimental results show that in-cell routing is successful for all cells with solutions with 2 more CPPs. With this step, Csyn-fp is able to provide diverse cell layouts, from which designers can choose the most suitable layout to their purpose.

2.3.6 Comparison with Previous Works

Table 2.7 shows the comparison of the average number of CPPs (i.e. cell size) and its running time produced by Csyn-fp and the SMT-based transistor placement algorithm in [2]. The main obstacle of the SMT-based approach is the runtime scalability[62]. As the cell size and the number of transistors increase, the time spent by SMT-based method increases exponentially. As a result, a complete solution is not obtainable in a reasonable time for large cell. In fact, for large cells of the ASAP 7nm library, the SMT-based approach in [2] could not find placement solutions within 2 hours. Thus, only the cells whose layouts were obtained within 2 hours are compared with ours.

The comparison shows that Csyn-fp generates cell layouts of smaller area over the layouts produced by [2], which is because the SMT-based method performs static folding in order to avoid explosive runtime. Nevertheless, total runtime of Csyn-fp is only 13.1% of that spent in [2]. The runtime saving is mainly contributed by our speeding up techniques, which efficiently prune the search space while maintaining the area optimality.

Table 2.7: Comparison of CSyn-fp with the SMT-based placement method [2] for netlists of the cells in ASAP 7nm library.

Cells		SMT-based [2]		CSyn-fp	
Type	#Cells	#CPPs	Runtime (s)	#CPPs	Runtime (s)
AO/AOI	38	8.9	43.060	8.8	9.232
OA/OAI	30	8.7	15.345	8.7	3.401
AND/NAND	16	7.1	0.964	7.1	0.066
OR/NOR	15	7.8	31.007	7.8	0.067
XOR/XNOR	6	10.7	11.821	10.7	1.460
BUF/INV	25	9.2	13.330	9.2	0.040
HA	5	6.2	0.270	6.2	0.056
Latch	1	15.0	506.422	15.0	0.036
		1.000	1.000	**0.996**	**0.131**

Chapter 3

Pin Accessibility and Routing Congestion Aware DRC Hotspot Prediction using Graph Neural Network and U-Net

3.1 Preliminary

3.1.1 Graph Neural Network

GNN is a powerful deep ML model specialized for graph data. Since many EDA problems can be formulated into optimization problems on graphs, GNN has been widely adopted in this field. (e.g. [37, 43, 63, 64, 65, 66, 67, 68])

A graph $G(V, E)$ is defined as a set, V, of vertices and a set, E, of edges. A vertex matrix $V \in \mathbb{R}^{n \times d}$ is a two-dimensional matrix, in which $n = |V|$ and d is the size of initial feature vector of each node in V. The edges in E are represented by an adjacency matrix $A \in \{0, 1\}^{n \times n}$. A GNN takes $G(V, E)$ with A as input and generates the embedding vector of every node in G. Similar to the convolutional layer of CNN, a graph convolutional layer is a main component of GNN, which iteratively updates node features considering the influence of the neighboring nodes.

The process on a graph convolutional layer consists of two main operations, which are *message generation* and *neighbor aggregation*. Fig. 3.1 illustrates the operation at

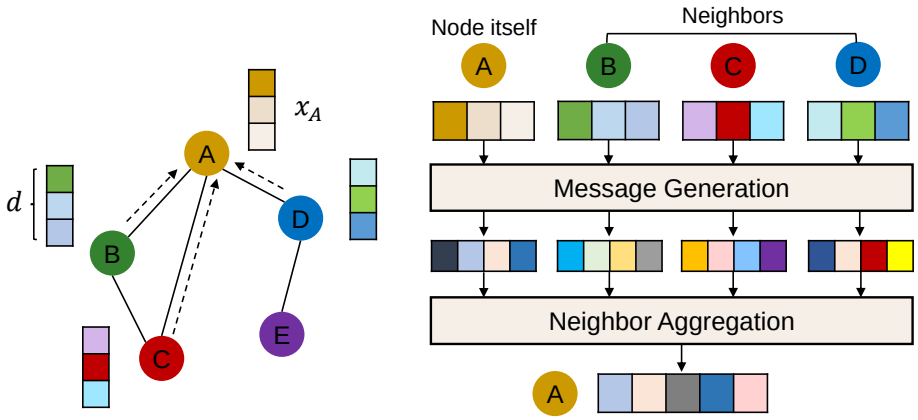


Figure 3.1: Illustration of processing on a single graph convolutional layer in graph neural network.

a single graph convolutional layer. For each node, its neighboring nodes generate *messages* by applying learnable fully connected layers to its node feature. These messages are aggregated into one reduced message, and the feature of the node is updated by using the aggregated message. The graph convolutional layer operation can be expressed as follows [69]:

$$x_i^{(l)} = AGG^{(l)}(\{MSG^{(l)}(x_j^{(l-1)}),\ \ j \in \{N(i) \cup i\}\}) \tag{3.1}$$

where $x_i^{(l)}$ denotes a feature of node i at the l-th graph convolutional layer, $N(i)$ denotes the neighboring nodes of i, and $MSG^{(l)}$ and $AGG^{(l)}$ indicate the message generation and neighbor aggregation operation in the l-th graph convolutional layer. $AGG^{(l)}$ is typically a sum, max or mean operation. Existing GNNs such as GCN [69], GraphSAGE [70] and GAT [71] differ with how graph convolutions are performed. The global pooling layer is often applied after completing the operations on GNN to obtain an embedding vector of the entire graph by aggregating the final features of all nodes in the graph.

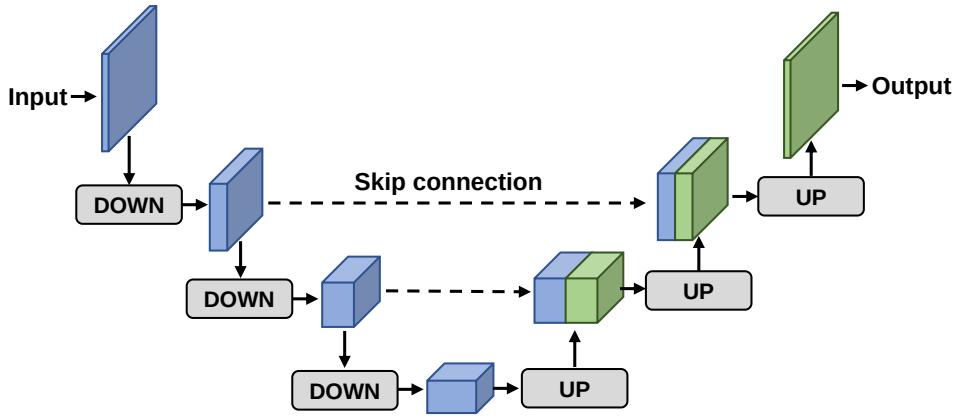


Figure 3.2: Illustration of conventional U-Net architecture [4].

3.1.2 Fully Convolutional Network

Fully convolutional network (FCN) [72] is proposed to solve semantic segmentation task, and all layers are composed of convolutional layers. Semantic segmentation in image recognition is to identify the class of every pixel in an input image. Thus, ML model for semantic segmentation should be able to take an arbitrary size of input images and outputs images of the same size as that of the inputs. DRC hotspot prediction problem can be seen as semantic segmentation if a kind of grid-based prediction is tried.

U-net [4] is an FCN based network and has achieved a great success in semantic segmentation. It adopts encoder-decoder structure like that shown in Fig. 3.2. Encoder gradually down-sample the input by applying a series of convolutional layers and pooling layers. Decoder is also comprised of the multiple repetitive up-sampling units, which takes a compacted feature map from the encoder as input, restoring them into the original size of input. Each up-sampling unit first receives the intermediate feature map from the encoder on the same level through the *skip connections*, and concatenates it with the output of the previous up-sampling unit. It typically applies transposed convolutional layers to scale up the input. The skip connection plays an

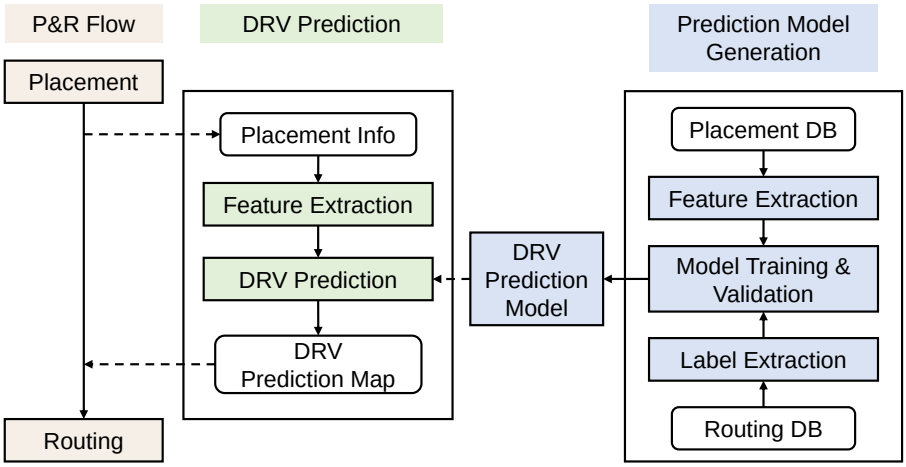


Figure 3.3: The overall flow of our proposed methodology combined with the conventional P&R flow.

important role in U-net, enabling the network to generate a high-quality prediction.

3.2 Proposed Prediction Methodology

3.2.1 Overall Flow

The overall flow of our proposed prediction methodology is depicted in Fig. 3.3. First, the entire placement region is divided into two-dimensional ($W \times H$) grids whose width and height are equal to that of G-cell. For the training data preparation, we conducted placement and routing on reference circuits with various placement settings. Input features are extracted at the placement and DRVs are extracted as the ground-truth after detailed routing. Output label is also a two-dimensional binary map with the same size of input feature map, indicating which grids are DRC hotspot.

Trained model can be adopted in the commercial place-and-route flow. After placement, input features are extracted from the placement results, and DRC hotspot prediction is generated by our prediction model, which can be utilized to optimize the

placement before routing.

3.2.2 Pin Proximity Graph

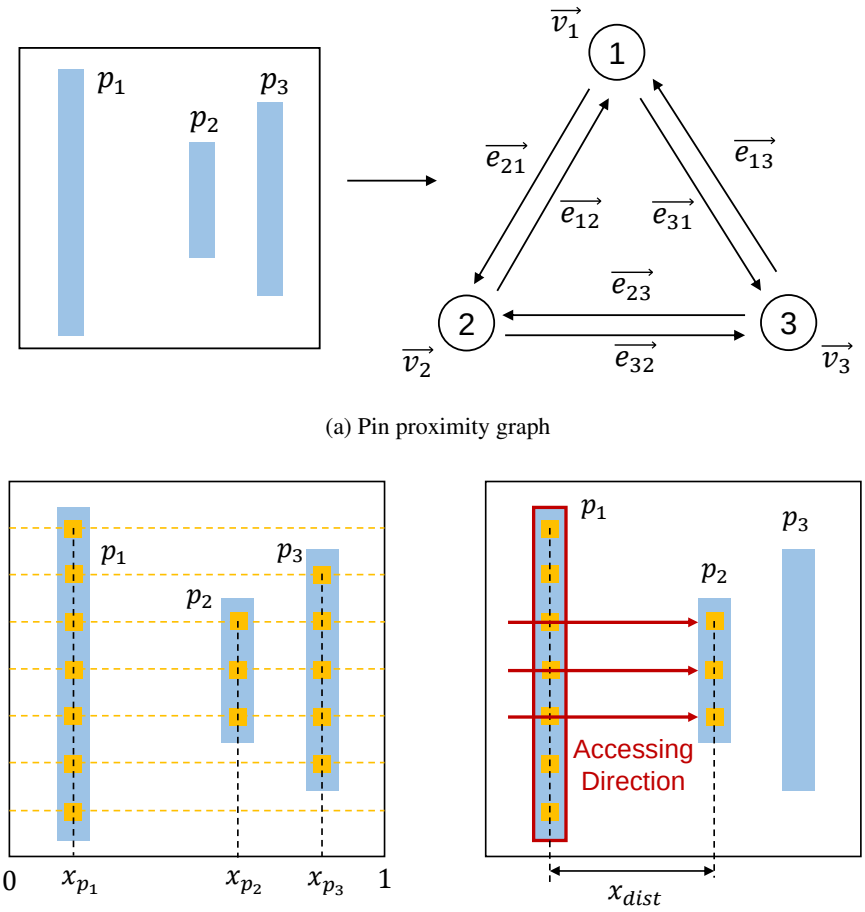
There are four factors that affect pin accessibility; (1) shape (or length) of pins, (2) approaching direction of nets to connect pins, (3) pin accessing disturbance caused by neighboring pins, and (4) number of nets passing through the grid. To accommodate this information, we formulate the pin information in a local grid as *pin proximity graph* where each node indicates a distinct pin and an influence between two pins is represented by the existence of edge between the two nodes corresponding to the pins.

For each grid $g_{(x,y)}$, suppose that there is a set of pins $P = \{p_1, p_2, ..., p_i\}$ in $g_{(x,y)}$. In pin proximity graph $\mathcal{G}_{(x,y)}(V, E)$, every node $v_i \in V$ corresponds to a distinct pin $p_i \in P$, and the node feature of v_i is formulated into $\overrightarrow{v_i} = (x_{p_i}, \overrightarrow{d_{p_i}})$ whose details are as follows:

- **Average x-coordinate pin access point x_{p_i} :** This feature is the average value of the x-coordinates of all access points in pin p_i. In this case, the x-coordinate is a relative coordinate based on grids. For example, if the access point is located on the left edge of the grid, the x-coordinate is set to 0, and if it is located on the right edge, the x-coordinate is set to 1. This feature indicates the x-directional location of p_i.

- **Dominate x-coordinate pin access point dx_{p_i} :** This feature is the most frequent value of the x-coordinates of all access points in pin p_i. In advanced nodes, as the height of standard cell decreases, most pins have polygon shapes. In order to express the shape of the pin in more detail, and since pin p_i has the highest probability of being accessed through an access point with x-coordinates of dx_{p_i}, so this value is incorporated in the node features.

- **Pin digit vector $\overrightarrow{d_{p_i}}$:** The more the number of access points on a pin that span metal 2 tracks is, the better the pin accessibility is. Pin digit vector of pin p_i is a



(a) Pin proximity graph

(b) Node feature

(c) Edge feature

Figure 3.4: Overall picture of pin proximity graph.

binary vector that indicates if the individual metal 2 tracks cross an access point on p_i or not. For example, the cell in Fig. 3.4(a) has three pins p_1, p_2, and p_3 in the grid. The yellow dotted lines indicate metal 2 tracks, and there are 7 metal 2 tracks inside the grid, in which p_3 does not have an access point on both top and bottom metal 2 tracks, thus, $\overrightarrow{d_{p_3}} = [0\ 1\ 1\ 1\ 1\ 1\ 0]$.

Edge $e_{ij} \in E$ is a directed edge from $n_j \in V$ to $n_i \in V$ and indicates a disturbance caused by the existence of p_j when accessing p_i. If pins p_i and p_j are located within a certain short distance, which implies the two pins involve a nontrivial amount of

interaction, e_{ij} exists. We extract diverse features that affect pin accessibility between two pins, and formulate the features into $\overrightarrow{e_{ij}} = (\overrightarrow{d_{ij}}, x_{dist}, po_{ij}, h_d)$ whose details are as follows:

- **Relative position of p_j with respect to accessing direction ($\overrightarrow{d_{ij}}$) of p_i :** For pin p_i, we apply the method presented in [49] to estimate the approaching direction of the nets among four possible directions (left, right, up, down), in which it uses FLUTE [73] and edge shifting [74] techniques for fast wirelength and congestion driven Steiner tree generation. We call the estimated direction *primary* direction, the opposite *last* direction, and the rest *secondary* direction. If p_j locates in the primary direction, it greatly affects the p_i accessibility. $\overrightarrow{d_{ij}}$ is a one-hot vector that represents the location of p_j with respect to p_i and its approaching direction. For example, in Fig. 3.4(c), FLUTE estimates that p_2 is accessed from the left direction. Since p_1 is located on the left of p_2 (i.e. primary direction), $\overrightarrow{d_{21}} = [1\ 0\ 0]$. Otherwise, p_3 is located on the right of p_2 (i.e. last direction), $\overrightarrow{d_{23}} = [0\ 0\ 1]$.

- **Distance (x_{dist}) between p_i and p_j :** The closer two pins are located, the greater the amount of influence on each other is. Especially, end-of-line or spacing rule violation can possibly occur if the two pins are closely located.

- **Overlapped access point track ratio po_{ij} :** This feature refers to the ratio of the number of metal-2 tracks on which both p_i and p_j have access points to the number of metal-2 tracks on which p_i has access points. For example, pin p_1 in Fig. 3.4(b) has access points on 7 metal-2 tracks, and both pin p_1 and p_2 has access points on 3 metal-2 tracks in the center, from which $po_{12} = 3/7$. This feature represents the amount of horizontal influence of p_j on p_i.

- **Horizontal routing congestion on grid h_d :** A high horizontal routing congestion on the grid implies a high difficulty in accessing the pins in the grid due to the influence of nets passing through the grid. Hence, we adopt horizontal

net density as a feature of the edge of the pin proximity graph. The detailed calculation of horizontal net density will be described in Sec. 3.2.3.

With these features, pin proximity graph can efficiently represent the complex interaction among the pins in the grid. Each grid generates an independent pin proximity graph, and there is no external connection between two graphs that correspond to two distinct grids. We transform the graphs of all grids into a three-dimensional pin proximity graph map while preserving location information.

3.2.3 Grid-based Features

The performance of prediction model is greatly affected by the features extracted from the grid. Previous works [50, 51, 53, 55, 56, 57, 58] have proposed various kinds of grid-based features. Through intensive experiments with many input features, we chose a set of representative features that significantly affect the model performance. Those features are described below.

- **Pin density:** This feature indicates the ratio of the area occupied by pins to the area of the grid. Since most pins are on metal 1 layer, the density is extracted individually for metal 1 pins and metal 2 pins. A high value of pin density implies an existence of a large number of pins on the grid, which leads to cause a dense routing congestion around the grid.

- **Global/Local net:** Global net is an incoming or outgoing edge that connects a pin inside the grid, and local net connects pins inside the grid. We count the number of these nets and use them as input feature. A large number of those nets on the grid means a high possibility of occurring DRV.

- **Long/Short RUDY:** RUDY [5] is a fast routing congestion estimator from a cell placement result. It assumes that net wire is equally distributed in its bounding box. Given bounding box of net n of size $w \times h$ grid, RUDY of n is defined as

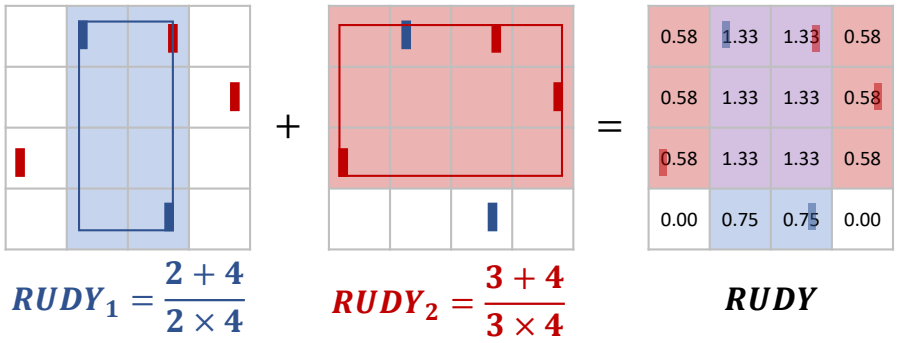


Figure 3.5: Example of calculating RUDY [5] for each grid.

the expected number of nets passing through the unit grid inside the bounding box, that is, $RUDY_n = \frac{w+h}{w\times h}$. This value is added to every grid overlapped with the bounding box. We generate two different RUDYs: Long RUDY for nets with a large bounding box and short RUDY for the short nets. If the half parameter of bounding box of a net is longer than 15 grids, it is classified as long net.

- **Horizontal/Vertical routing capacity:** If there are many metal tracks available, routing can be successfully performed without generating DRV even in areas with high routing demand. Since routing is performed in the horizontal and vertical directions, we use the total number of horizontal/vertical metal tracks per grid for each direction as feature.

- **Horizontal/Vertical net density:** Similar to RUDY, it indicates the expected number of nets passing in the horizontal/vertical direction, which was firstly proposed in [75]. Given bounding box of net n of size $w \times h$ grid, horizontal density is $\frac{1}{w}$ and vertical density is $\frac{1}{h}$ for unit grids of each row and column. These values are added to the grid overlapped with the bounding box.

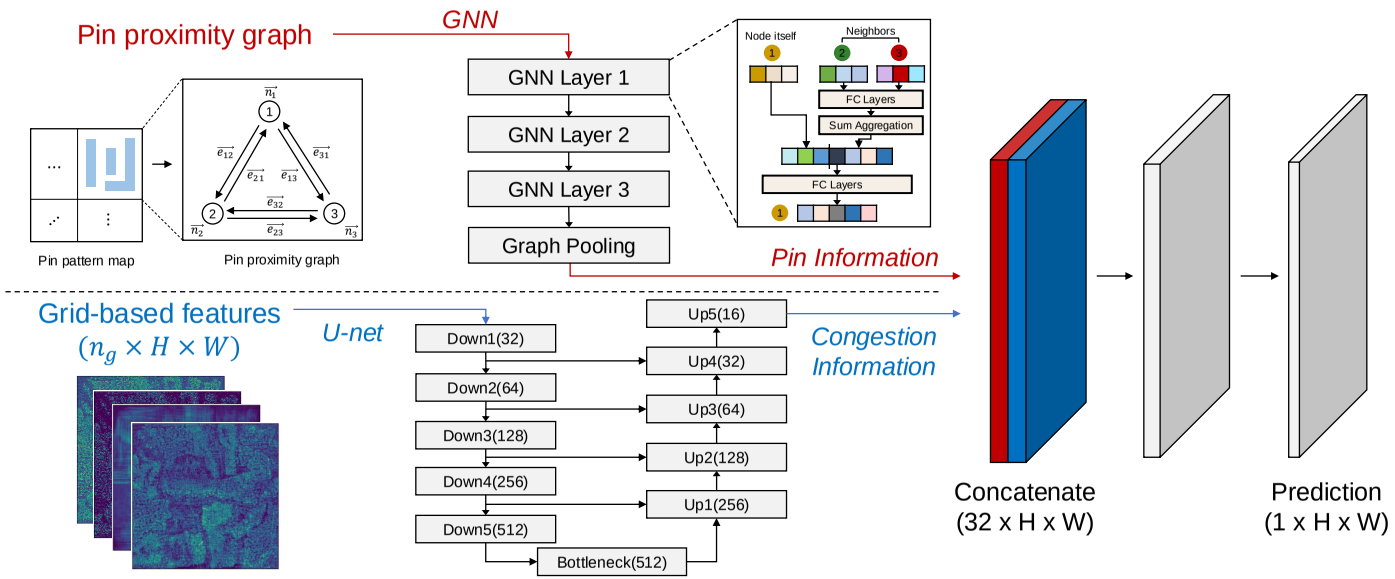


Figure 3.6: The overall architecture of our proposed DRC hotspot model PGNN.

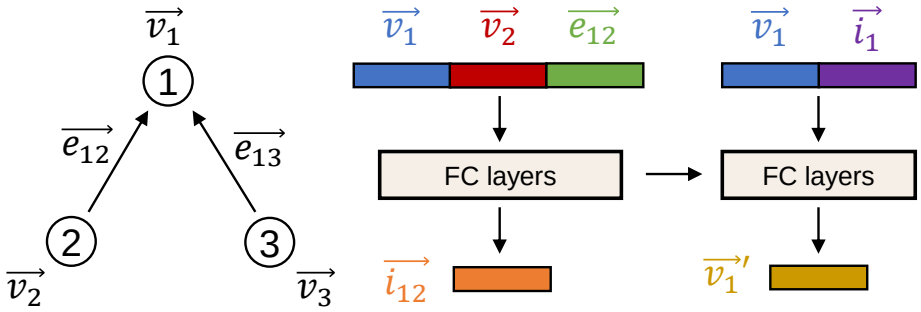


Figure 3.7: Overall picture of the proposed single graph convolutional layer in PGNN.

3.2.4 Overall Architecture of PGNN

Our proposed model called PGNN is composed of three modules: GNN module for modeling pin information inside grid, U-net module for modeling routing congestion, and final prediction module that accepts the outputs of the two modules and produces prediction. Fig. 3.6 shows our PGNN architecture. The GNN and U-Net modules accommodate different types of input feature. The input feature of GNN is the pin proximity graph, which captures local pin accessibility inside grid. On the other hand, U-net takes a set of grid-based features as input and generates representation of each grid while preserves routing congestion from the global point of view. Then, the outputs of the two modules are concatenated, and the final prediction module produces DRC hotspot prediction map. If the prediction value exceeds a given threshold, we report the corresponding grid to be DRC hotspot.

3.2.5 GNN Architecture in PGNN

Since our pin proximity graph has edge features, conventional GNNs such as GCN [69], GraphSAGE [70] and GAT [71] are not able to be directly applied to our graph. Therefore, we develop a novel graph neural network architecture that is highly applicable to our graph.

In our problem, the node features obtained from GNN express the accessibility of

the pins representing the corresponding nodes. Our GNN layer iteratively updates the node features by considering the information of the neighboring nodes. Fig. 3.7 shows the process of updating the node features of the graph in a single GNN layer. Note that each node j represents pin p_j in the grid. For updating the feature of node j (e.g., $j = 1$ in Fig. 3.7), GNN layer first calculates $\overrightarrow{i_{jk}}$ which is the disturbance of the neighboring pin p_k when accessing pin p_j. Since $\overrightarrow{i_{jk}}$ is affected by edge feature as well as node feature, it can be calculated as follows:

$$\overrightarrow{i_{jk}} = f_\theta(\overrightarrow{v_j} \| \overrightarrow{v_k} \| \overrightarrow{e_{jk}}) \tag{3.2}$$

where f_θ is a set of learnable parameters of fully-connected layers.

The total disturbance by all neighboring pins is the sum of all individual disturbance $\overrightarrow{i_{jk}}$:

$$\overrightarrow{i_j} = \sum_{k \in n(j)} \overrightarrow{i_{jk}} \tag{3.3}$$

where $n(j)$ is the set of neighboring nodes of node j.

Finally, the node feature $\overrightarrow{v_j}$ is updated through the fully-connected layer after concatenating the original node feature and the total disturbance $\overrightarrow{i_j}$:

$$\overrightarrow{v_j}' = f_\phi(\overrightarrow{v_j} \| \overrightarrow{i_j}) \tag{3.4}$$

where f_ϕ is a set of learnable parameters of fully-connected layers.

For the pin proximity graph of each grid, the node features are obtained by processing three consecutive GNN layers. To process them efficiently, we express the graphs of all grids as a single super-graph, so that parallel processing can be applied. To conduct grid-based prediction, we transform all node features into global graph feature by applying graph average pooling, which simply takes an average of all node features. We then convert this graph feature vector generated for each grid into a three-dimensional matrix to restore spatial information.

3.2.6 U-net Architecture in PGNN

Fig. 3.8 shows the detailed architecture of U-Net in PGNN. The number in parenthesis in each block of Fig. 3.8 indicates the number of output kernels. The input of U-net is grid-based features described in Section 3.2.3 with the size of $10 \times H \times W$ where 10 is the number of grid-based features. For the encoding path of the U-net, we adopt ResNet [76] backbone architecture, which consists of five down-sampling blocks adopting shortcut connection structure. Each block applies a pair of 3x3 convolutional layers to its input. The first convolutional layer uses stride 2 to halve the size of input. Shortcut connection applies 1x1 convolutions with stride 2 to control the output dimension. Therefore, the encoder reduces the input size by 32 times.

Subsequently, the bottleneck block in Fig. 3.8 updates the output of the encoder by 3x3 convolutions, and then the decoder restores the compressed feature map to the original size through processing five up-sampling blocks. Each up-sampling block doubles the input size by using a transposed convolutional layer, and concatenates the doubled input with the feature map produced by the down-sampling block in the same block level (counting from the bottleneck block) of the up-sampling block, followed by performing 3x3 convolutions. We adopt group normalization [77] and ReLU activation layers in each block. The output size of the decoder is $16 \times H \times W$.

3.2.7 Final Prediction in PGNN

Our final prediction module concatenates the output feature maps of GNN and U-net, resulting in a feature size of $32 \times H \times W$. The module then processes two 3x3 convolutional layers, one with 16 kernels and the other with 1 kernel, to generate a final prediction map of size $H \times W$. Finally, the sigmoid layer is applied to make [0, 1] probability score. If the score exceeds a given threshold, the corresponding grid is classified as DRC hotspot grid.

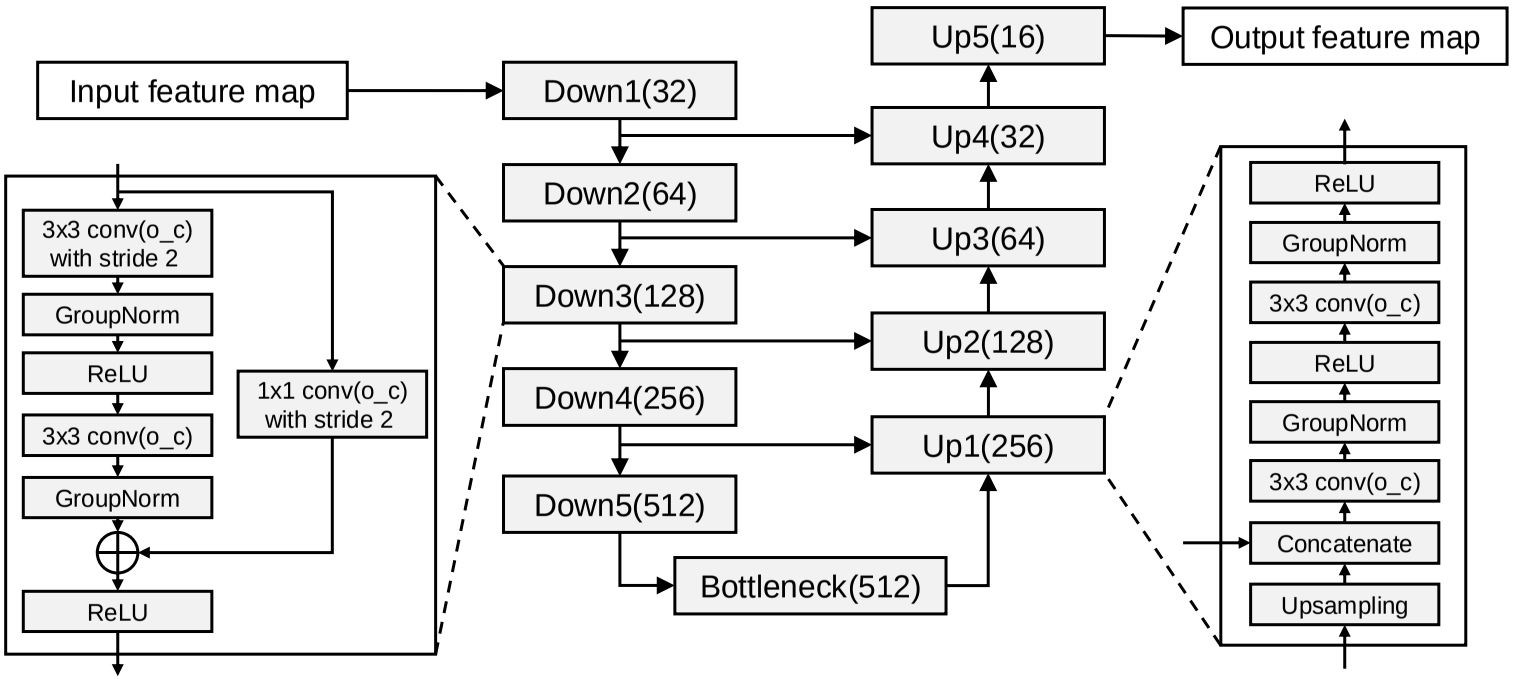


Figure 3.8: Block-level detailed architecture of the U-net module in PGNN.

3.3 Experimental Results

3.3.1 Experimental Setup

We implemented the feature extractor in C++ and PGNN in *Python* with *Pytorch* [78] and *Pytorch Geometric* library [79]. We generated training and test data from 10 Opencore [80] circuits. Table 3.1 shows the statistics of the benchmarks used in our experiments. We obtained a total of 604 different placement data with different placement parameter settings such as chip utilization, clock period, and number of routing layers. With Nangate 15nm library [81], we repeatedly applied logic synthesis, clock tree synthesis, placement, and routing to collect ground-truth labels. We used *Synopsys Design Compiler* for synthesizing all benchmarks, and used *Synopsys IC Compiler 2* for placement and routing. Experiments were conducted on a linux machine with Intel i7-9700K 3.6GHz CPU, 32GB memory and RTX 2080Ti GPU. The size of unit grid for prediction was $768nm \times 768nm$ which is equal to the height of standard cell. We trained PGNN by using AdamW optimizer [82] with initial learning rate 0.001 and weight decay 0.01. Cosine annealing scheduler [83] was utilized to control the learning rate. Each model was trained with 150 training iterations.

We used two different schemes to separate training and test set data, as follows.

- **Seen setting:** We randomly divide the total of 604 placement data produced from 10 benchmark circuits into 10 groups. Then, we conduct 10-fold cross-validations on the groups i.e., at each fold, 9 groups are used for training and the remaining 1 group for test. That is, we repeat experiments 10 times, each using a distinct group for test and the rest for training. In this setting, both the training and test sets may include data generated from the same benchmark circuit.

- **Unseen setting:** We use the data from 9 benchmark circuits among 10 for training and the remaining 1 for test. We repeat experiments 10 times, each using the data from a distinct benchmark circuit for test and the rest for training. This setting ensures that the test circuit is completely unseen from the training data.

Table 3.1: Statistics for the circuits from the Opencore designs used in experiments.

Designs	**#Gates**	**#Nets**	**#Grids**	**#Placements**
AES_128	124,699	115,518	83,507	60
B18	38,166	33,556	21,267	60
B19	75,145	66,499	43,687	58
ECG	148,039	125,325	114,893	60
ETH	45,910	46,012	56,665	43
JPEG	291,460	244,078	164,686	74
LDPC	57,956	76,921	40,549	60
NOVA	208,536	152,845	121,483	72
TATE	314,515	256,753	222,333	57
VGA_LCD	118,608	76,464	115,718	60

In both settings, 10% of training set is used as a validation set to determine the classification threshold.

Table 3.2: Illustration of the confusion matrix.

		Actual	
		Positive	Negative
Prediction	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

In binary classification problem, prediction results are classified into four groups i.e., true positive (TP), true negative (TN), false positive (FP), and false negative (FN) according to its prediction and ground-truth. TP/TN are the positive/negative samples that predict correctly, and FN/FP are the positive/negative samples that mispredict.

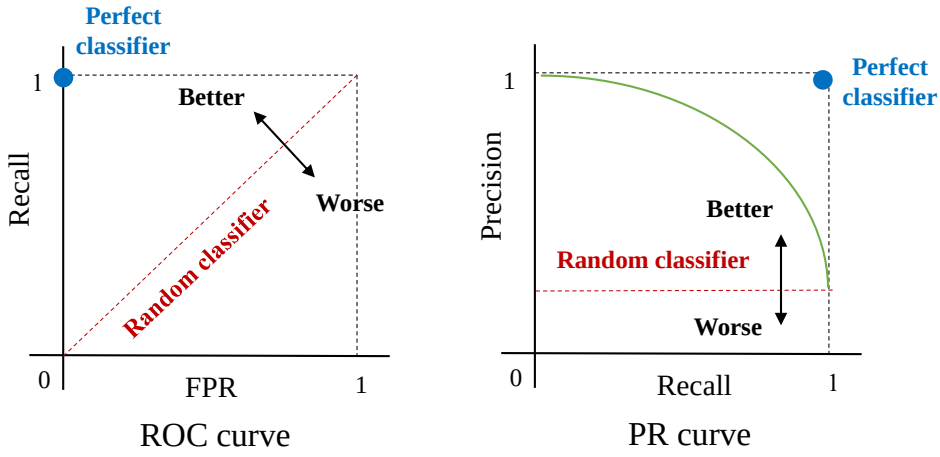


Figure 3.9: Illustration of ROC(Receiver Operating Characteristic) curve and PR(Precision and Recall) curve.

(Confusion matrix is shown in Table 3.2.) Based on the four groups, we use the following five metrics to evaluate the performance of the model in experiments.

- **Accuracy** $= \frac{TP+TN}{TP+FP+FN+TN}$
- **Precision** $= \frac{TP}{TP+FP}$
- **Recall** $= \frac{TP}{TP+FN}$
- **FPR (false positive rate)** $= \frac{FP}{FP+TN}$
- **F1-score** $= 2 \times \frac{Precision \times Recall}{Precision+Recall}$
- **AUC metrics** : DRV prediction model outputs a 0 to 1 continuous prediction value, which indicates the probability of DRV occurrence inside the grid. At this time, the performance of the model highly depends on how the threshold is selected. To exclude the impact of threshold selection and examine the pure performance of the model, we used AUC (Area Under Curve) metrics such as AUC-ROC and AUC-PR. ROC curve is a graph with FPR and recall on the x-axis and y-axis when the threshold changes from 0 to 1, and PR curve has recall

and precision on the x-axis and y-axis with the same setting as ROC curve. Fig 3.9 shows an illustration of ROC curve and PR curve. AUC-ROC and AUC-PR are an area below the ROC curve and PR curve and their range is 0 to 1. High values indicate that the model has high predictive performance, and the random classifier shows a 0.5 value of these.

Models with better performance shows higher accuracy, precision, recall, and F1-score with lower FPR. Especially in DRC hotspot prediction problem, since our dataset is highly imbalanced i.e., only 2.4% of grids are DRC hotspot, F1-score is the most reliable metric for evaluating the model performance.

3.3.2 Analysis on PGNN Performance

Table 3.3 summarizes the F1-scores of our PGNN produced by varying the usage of input features. The GNN in PGNN is used to model pin accessibility in each local grid, and the U-Net in PGNN is used to model routing congestion considering the information of neighboring grids.

Table 3.3: Performance analysis, in terms of F1-score in seen data setting, of our PGNN by varying the usage of input features.

Module	**Features**	**Model setting**						
GNN	Pin proximity graph	✓						✓
U-Net	Pin density		✓	✓	✓	✓	✓	✓
	Global/Local net			✓	✓	✓	✓	✓
	Routing capacity				✓	✓	✓	✓
	Long/Short RUDY					✓	✓	✓
	H/V net density						✓	✓
F1-score (%)		**54.59**	46.80	48.26	55.18	64.67	**66.66**	**71.91**

Firstly, We conduct experiments on the effectiveness of grid-based features by selectively including the features to the U-Net. It is seen from Table 3.3 that pin density,

routing capacity, and RUDY feature play a key role on the U-Net performance. In the routing congestion perspective, a high pin density in a grid implies a large number of pins, which indicates a large number of nets are likely to be derived from the grid. Although global/local net feature tells the exact number of nets on the grid, pin density feature seems to sufficiently infer this information. On the other hand, RUDY feature provides a coarse-grained routing congestion estimation, revealing a high leap of F1-score if included to the U-net. For H/V net density feature, though it estimates congestion in horizontal and vertical directions, the calculation method is similar to RUDY. Consequently, it reveals little performance improvement. Finally, for routing capacity feature, it helps to distinguish the situations of DRV variations by the number of routing layers used in the same placement.

In summary, Table 3.3 shows that GNN and U-Net alone achieve 54.59% and 66.66% F1-score whereas the combined model achieves 71.91%. This means the prior works that have used pin density feature only for describing pin accessibility do not work well. By devising the pin proximity graph, formulating it into GNN, and integrating the GNN with U-Net, we are able to achieve a high F1-score of DRC hotspot prediction.

3.3.3 Comparison with Previous Works

To assess the performance of PGNN, we implement a set of representative DRV prediction models from existing studies. It includes global routing congestion (denoted as GR-Cong) from the representative commercial EDA tool, and representative research results, such as RouteNet [55] and J-Net [58]. GR-Cong is obtained from *ICC2* after global routing stage, and grids with high routing congestion are classified as DRC hotspot. To implement J-Net similar to [58], pin pattern of the grid is represented by 32×32 pixels, and input feature maps are cropped with 80×80 size of window. Thus, the size of pin pattern image for input becomes 2560×2560. The model architecture and features of RouteNet and J-Net used in our experiment are the same as that in [55]

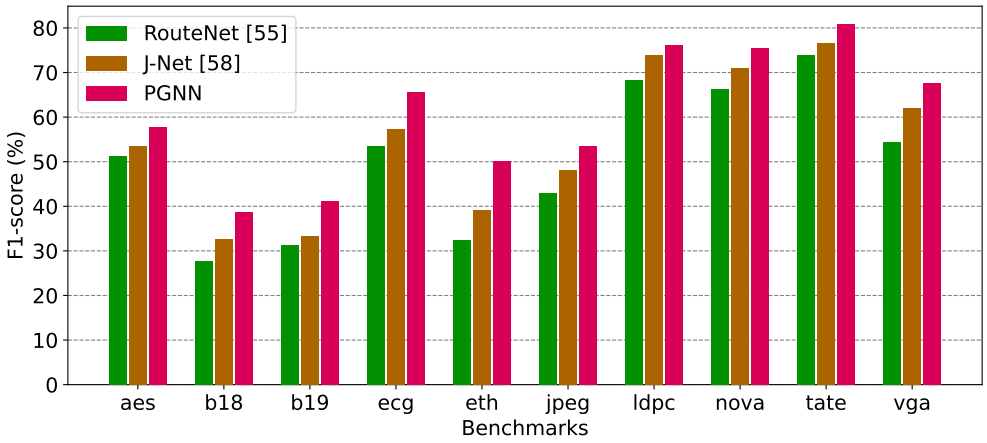


Figure 3.10: F1-score comparison of RouteNet, J-Net and PGNN for all ten benchmarks in seen setting.

and [58].

• **Comparing model performance**: Table 3.4 summarizes the comparison of model performance of PGNN with that of the previous works. PGNN shows superior performance over other models even without using GR congestion as a feature, achieving 7.8%, 12.5% of improvements on F1-score over J-Net in seen and unseen settings, respectively. Figs. 3.10 and 3.11 show the comparison of F1-scores of RouteNet, J-Net and PGNN on individual circuits in both settings. It is shown that PGNN outperforms other models on all 10 benchmarks. Table 3.5 and 3.6 indicates detailed comparison of precision, recall, FPR and F1-scores with previous works. For all benchmarks, PGNN shows the highest prediction performance.

One main reason of the inferior performance of RouteNet is that it does not consider pin accessibility since it just uses grid-based features. J-Net expresses pin information as pin pattern image. However, pin pattern image alone is not sufficient to provide detailed information on how each net accesses to its target pins. Furthermore, since pin pattern image requires a massive amount of parameters, J-Net should perform input cropping to mitigate GPU memory overhead for inferencing, losing neighboring

Table 3.4: Comparison of the performance of DRC hotspot prediction in seen and unseen setting for all ten benchmarks. GR-Cong indicates global routing congestion-based DRC hotspot prediction.

Model	Use GR?	Seen setting					Unseen setting				
		Accuracy	Precision	Recall	FPR	F1-score	Accuracy	Precision	Recall	FPR	F1-score
GR-Cong	Yes	97.50%	40.39%	54.39%	1.53%	43.28%	97.50%	40.39%	54.39%	1.53%	43.28%
RouteNet [55]	Yes	98.28%	62.39%	62.92%	0.89%	62.65%	98.08%	59.16%	52.62%	0.85%	55.70%
J-Net [58]	No	98.42%	66.61%	66.77%	0.81%	66.69%	97.67%	61.67%	48.42%	0.88%	57.48%
PGNN	No	98.70%	71.28%	72.55%	0.69%	**71.91%**	98.18%	58.21%	72.80%	1.23%	**64.69%**

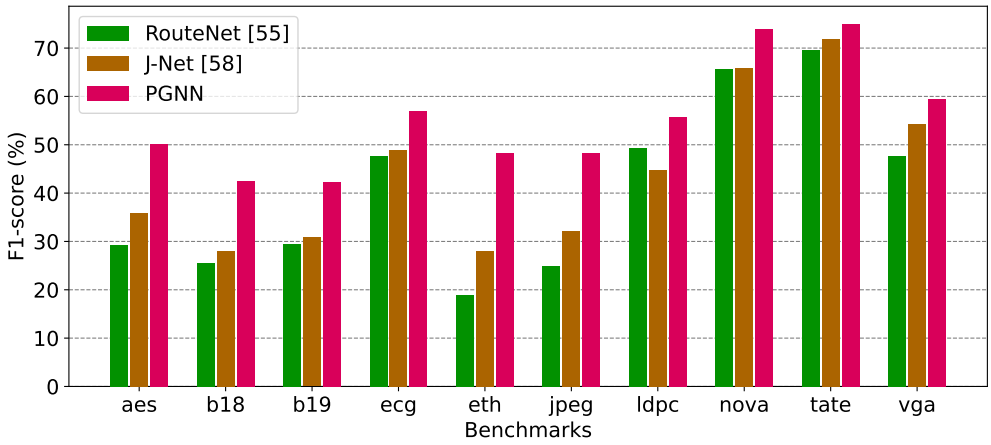


Figure 3.11: F1-score comparison of RouteNet, J-Net and PGNN for all ten benchmarks in unseen setting.

information which limits J-Net performance. On the other hand, the pin proximity graph of PGNN can effectively model not only the spatial information of pins but also information on the direction of the pin signals by using much fewer parameters than that of J-Net. In addition, our PGNN integrating GNN architecture abstracting pin proximity graph exhibits a superior performance to J-Net.

Fig. 3.12 visualizes the DRC hotspot prediction results of the models for circuit ECG. The left-top figure shows ground-truth whose yellow dots indicate DRC hotspot grid. Red boxes in Figs. 3.12(b) and (c) represent grid region that is misclassified as DRC hotspot grid. Since RouteNet and J-Net do not consider pin accessibility in depth, they have relatively more misclassified regions over that by our PGNN.

• **Comparing training and inference time**: Model training and inference time is another important factor for model adoption. Table 3.7 compares the training and inference time for the largest circuit TATE among the benchmarks.

The inference task consists of two stages: feature extraction and model prediction. Since the model prediction stage can be performed quickly by using GPU, the feature extraction stage accounts for the majority of total inference time. Note that

Table 3.5: Comparison of the performance of DRC hotspot prediction on seen setting for individual benchmarks.

Design	SVM [51]				RouteNet [55]				J-Net [58]				PGNN			
	Prec.	Recall	FPR	F1	Prec.	Recall	FPR	F1	Prec.	Recall	FPR	F1	Prec.	Recall	FPR	F1
AES_128	42.16%	18.24%	0.46%	25.46%	50.72%	51.64%	0.92%	51.18%	56.55%	50.54%	0.79%	53.38%	59.51%	55.90%	0.70%	**57.65%**
B18	30.04%	17.15%	0.30%	21.84%	32.26%	24.34%	0.39%	27.75%	37.66%	28.76%	0.38%	32.61%	49.31%	31.93%	0.25%	**38.76%**
B19	32.16%	14.61%	0.25%	20.09%	41.22%	25.34%	0.29%	31.38%	42.97%	27.14%	0.29%	33.27%	52.64%	33.75%	0.25%	**41.13%**
ECG	42.71%	18.16%	0.72%	25.49%	49.91%	57.63%	1.72%	53.50%	56.71%	57.96%	1.35%	57.33%	64.38%	66.65%	1.10%	**65.49%**
ETH	35.36%	12.99%	0.15%	19.00%	58.77%	22.31%	0.10%	32.34%	55.82%	30.03%	0.17%	39.05%	59.37%	43.35%	0.19%	**50.11%**
JPEG	47.04%	8.18%	0.05%	13.94%	49.77%	37.93%	0.21%	43.05%	53.57%	43.55%	0.21%	48.04%	59.89%	48.46%	0.18%	**53.57%**
LDPC	60.37%	31.62%	1.12%	41.51%	66.09%	70.77%	1.96%	68.35%	70.70%	77.13%	1.74%	73.78%	76.74%	75.72%	1.24%	**76.22%**
NOVA	67.20%	37.87%	0.55%	48.44%	62.75%	70.34%	1.23%	66.33%	67.28%	74.90%	1.08%	70.88%	73.43%	77.56%	0.83%	**75.43%**
TATE	71.01%	33.04%	0.48%	45.10%	72.34%	75.35%	1.03%	73.82%	74.88%	78.22%	0.94%	76.51%	77.22%	84.79%	0.89%	**80.83%**
VGA_LCD	58.18%	26.14%	0.53%	36.08%	62.14%	48.34%	0.83%	54.38%	64.85%	59.29%	0.94%	61.94%	68.36%	66.86%	0.87%	**67.60%**

Table 3.6: Comparison of the performance of DRC hotspot prediction on unseen setting for individual benchmarks.

Design	SVM [51]				RouteNet [55]				J-Net [58]				PGNN			
	Prec.	Recall	FPR	F1	Prec.	Recall	FPR	F1	Prec.	Recall	FPR	F1	Prec.	Recall	FPR	F1
AES_128	28.22%	37.50%	1.75%	32.21%	49.30%	20.72%	0.39%	29.18%	62.94%	25.16%	0.30%	35.95%	60.19%	42.99%	0.52%	**50.16%**
B18	30.07%	17.30%	0.31%	21.96%	31.88%	21.36%	0.35%	25.58%	41.14%	21.25%	0.24%	28.02%	36.44%	51.12%	0.68%	**42.55%**
B19	32.11%	15.36%	0.26%	20.78%	36.96%	24.52%	0.34%	29.48%	44.17%	23.66%	0.24%	30.82%	49.83%	36.79%	0.30%	**42.33%**
ECG	40.05%	18.70%	0.83%	25.49%	44.08%	52.04%	1.96%	47.73%	47.48%	50.26%	1.69%	48.83%	43.74%	81.94%	3.13%	**57.04%**
ETH	34.76%	13.50%	0.16%	19.45%	48.92%	11.73%	0.08%	18.92%	59.54%	8.29%	0.04%	14.55%	50.52%	46.24%	0.29%	**48.28%**
JPEG	44.96%	7.56%	0.05%	12.95%	71.67%	4.78%	0.01%	8.96%	52.97%	23.10%	0.11%	32.17%	38.83%	64.19%	0.55%	**48.39%**
LDPC	63.21%	22.21%	0.70%	32.87%	54.56%	44.97%	2.02%	49.31%	62.22%	35.06%	1.16%	44.85%	83.43%	41.83%	0.45%	**55.72%**
NOVA	69.06%	34.15%	0.45%	45.70%	65.29%	65.91%	1.03%	65.60%	76.89%	57.77%	0.52%	65.97%	77.50%	70.79%	0.61%	**73.99%**
TATE	73.23%	25.39%	0.33%	37.71%	72.26%	67.14%	0.92%	69.61%	75.42%	68.60%	0.80%	71.84%	63.72%	90.87%	1.84%	**74.91%**
VGA_LCD	55.41%	28.25%	0.64%	37.42%	48.34%	47.15%	1.42%	47.74%	61.67%	48.42%	0.88%	54.24%	52.24%	68.99%	1.78%	**59.46%**

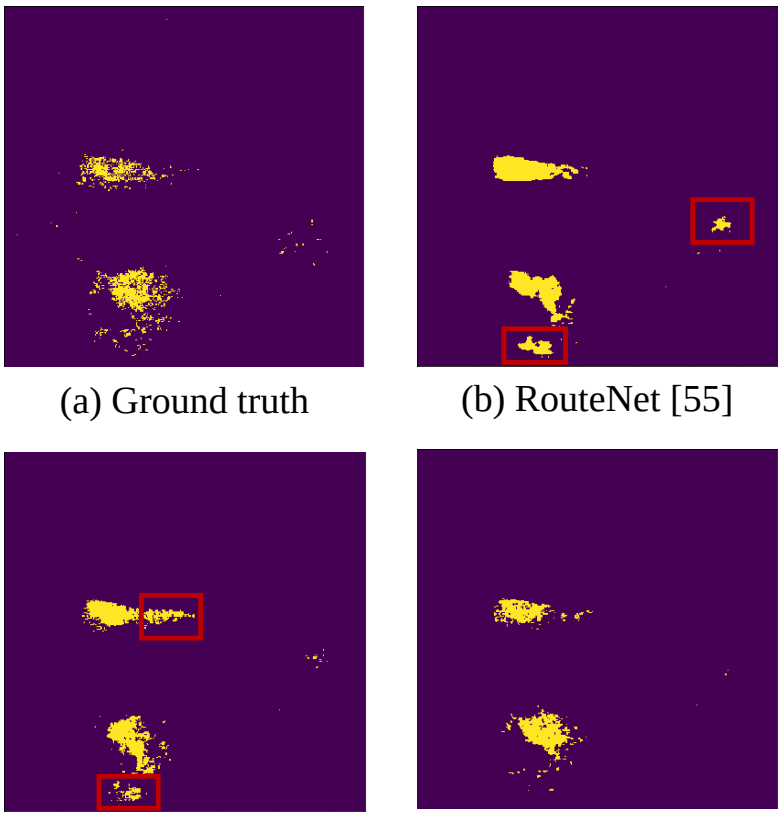


Figure 3.12: Visualization of DRC hotspot prediction. Yellow dots represent DRC hotspot grids.

Table 3.7: Comparison of the training and inference time of our PGNN for circuit TATE which has about 300,000 logic gates.

Task	**RouteNet [55]**	**J-Net [58]**	PGNN
Training Time	0.7 hours	31.7 hours	3.8 hours
Global routing	8.7 mins	N/A	N/A
Feature extraction	29.9s	640.0s	119.8s
Prediction	1.9s	12.5s	2.9s
Tot. Inference Time	9.2 mins	10.9 mins	**2.0 mins**

since RouteNet uses GR congestion as input feature, the time taken by global routing (8.7 mins) is included in the total inference time of RouteNet. Since RouteNet uses grid-based features only, its feature extraction time is the shortest. Besides grid-based features, J-net uses pin pattern image while PGNN uses pin proximity graph. For J-Net, pin pattern image requires a massive number of parameters and input cropping is essential, demanding expensive feature extraction time. For PGNN, generating pin proximity graph includes the execution of FLUTE [73] to estimate approaching direction and graph construction. (FLUTE is a look-up table based method to provide a fast runtime.) The inference time comparison in Table 3.7 shows that PGNN achieves 5.5× and 4.6× faster inference time than J-Net and RouteNet, respectively.

Regarding training time, early convolutional layers for J-Net to process pin pattern image require a huge amount of computation. PGNN can achieve 8.3× faster training time than J-Net by expressing core pin information using graph. Though model training is a one-time-only process, PGNN shows a better scalability over J-Net.

Fig 3.13 shows the inference time trend of RouteNet, J-Net and PGNN for all 10 benchmarks. All benchmarks are listed on the x-axis of the figure 3.13 in the order of the number of grids. For small-size circuits such as B18, B19 and ETH, the total inference time of the three models is not significantly different. However, in the case

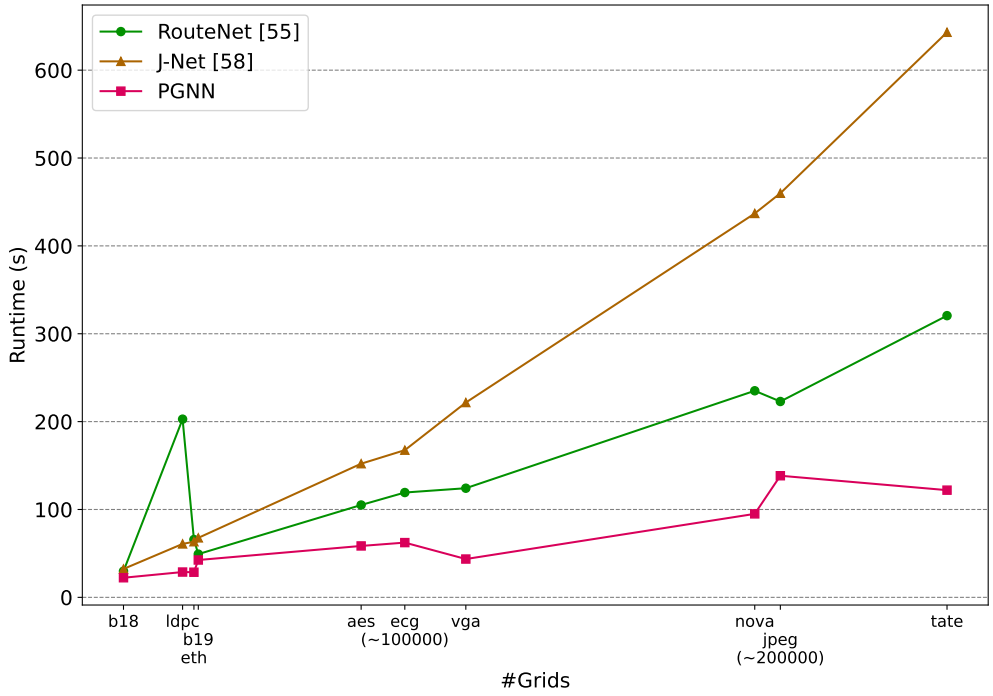


Figure 3.13: Inference time comparison of RouteNet, J-Net and PGNN for all ten benchmarks.

of LDPC, RouteNet shows a way longer runtime than the other models, which seems that LDPC has large pin-per-net ratio, resulting in a longer execution time of global routing. Also, as the number of grids in the benchmarks increases, J-Net and RouteNet shows a trend of rapid runtime increase than PGNN. Note that the total inference time of J-Net is dominated by the extraction time of pin pattern image and global routing time accounts for the majority of RouteNet's inference time. Since our graph model is able to express pin accessibility related information by using fewer parameters than pin pattern image and our prediction model does not require global routing information, PGNN shows better scalability than the other two models.

3.3.4 Adaptation to Real-world Designs

• **Adaptation to real-world size designs**: Benchmarks used in our experiments have less than 300,000 gates, which is smaller than the design used in the real-world industry which has at least 1 million gates. To verify the capability of our model on large-size designs, we first produced a set of test circuits, JPEG_X3 and TATE_X3, tightly linking three copies of original opencore circuit JPEG and TATE individually. Table 3.8 shows the placement and routing results of JPEG_X3 and TATE_X3. We used 70% and 1.0ns for utilization and clock period, and the two circuits have over 1 million cells after implementation.

Table 3.8: Statistics for the large-size circuits JPEG_X3 and TATE_X3 which are produced by combining three copies of the original circuit JPEG and TATE.

Circuit	Utilization	Clock period	#Cells	#Grids	#DRV	#DRV grids
JPEG_X3	70%	1.0ns	1,210,988	943,812 (971x972)	12,816	10,483 (1.11%)
TATE_X3	70%	1.0ns	1,012,885	738,440 (859x860)	175,118	97,709 (13.23%)

DRC hotspot prediction on a large-size circuit can not be obtained by a single run of PGNN due to the GPU memory limitation. To solve this, PGNN crops grid-based input feature map into the reasonable size of windows. The window size is 400×400 grids which is the maximum input size that PGNN can perform a prediction at once. The prediction is performed by sliding the window with 360×360 strides, and only the results of center 360×360 of the window are adopted as a prediction result to prevent the loss of surrounding neighborhood information.

Fig 3.14 shows a visualization of DRC hotspot prediction of JPEG_X3. Fig 3.14 (a) is a ground-truth result whose yellow dots indicate the actual DRC hotspot grid. Fig 3.14 (b) is DRC hotspot prediction obtained from PGNN, and the grids with brighter color has a higher probability of DRV occurrence. Finally, Fig 3.14 (c) is

Table 3.9: Step-by-step comparison of the total inference time of PGNN between the original circuits and the expanded circuits.

Task	JPEG	JPEG_X3	TATE	TATE_X3
DEF parsing	14.63s	65.24s	14.59s	50.54s
Grid-based feature extraction	9.78s	43.23s	10.60s	36.83s
Net direction estimation	24.90s	94.04s	26.29s	85.04s
Graph extraction	33.63s	126.56s	26.20s	101.98s
Input data generation	51.41s	189.27s	39.37s	149.00s
Prediction	2.36s	4.59s	2.90s	4.53s
Total inference time	2.28 mins	8.72 mins	2.00 mins	7.13 mins
	1.000	3.825	1.000	3.567

the DRC hotspot prediction after applying thresholding at Fig 3.14 (b). Although input feature cropping potentially loses neighborhood information of edge grids of the window, PGNN can generate high-quality prediction results by window overlapping.

Table 3.9 shows inference time comparison of JPEG_X3 and TATE_X3 with the original circuit JPEG and TATE. The inference time is composed of three stages: DEF parsing, feature extraction, and prediction. In the DEF parsing stage, cell placement information is extracted by the DEF format output of commercial P&R tools at the placement or clock tree synthesis stage. Then, the feature extraction stage obtains input features for PGNN from the placement information. It first extracts grid-based features for the input of U-Net, and for the extraction of the pin proximity graph, the net direction is firstly estimated using the FLUTE algorithm and the graph is generated based on it. Finally, extracted features are transformed into the proper data structure (npz format) compatible with the prediction model, and PGNN generates a DRC hotspot prediction map.

Experimental results show that JPEG_X3 and TATE_X3 have 3.825x and 3.567x longer inference time than JPEG and TATE respectively. Considering that JPEG_X3 and

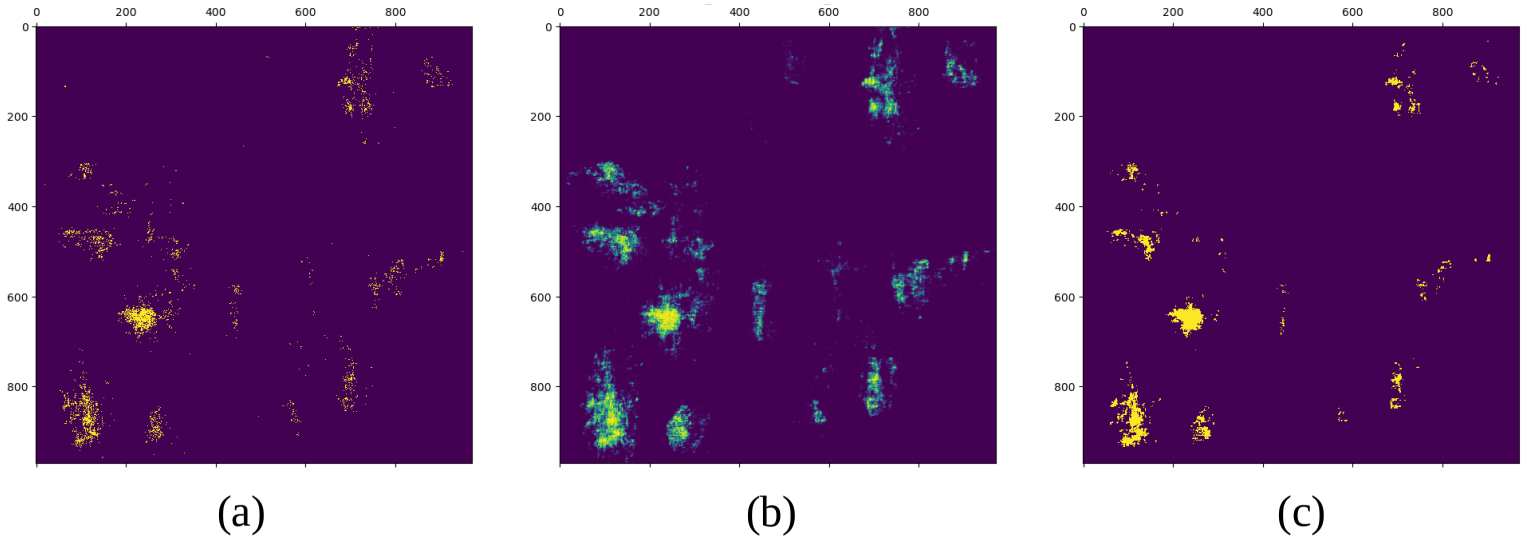
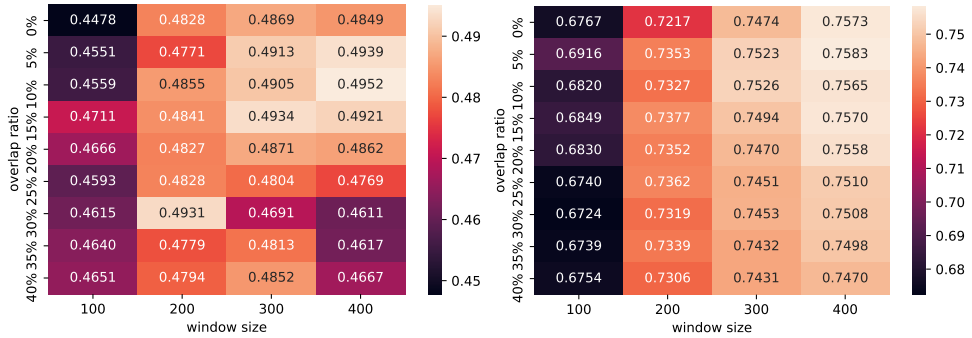


Figure 3.14: Visualization of DRC hotspot prediction of JPEG_X3. (a) Ground-truth. (b) DRC hotspot prediction logits. (c) DRC hotspot prediction after thresholding.



(a) JPEG_X3 (b) TATE_X3

Figure 3.15: F1-score according to the window size and overlap ratio when performing input cropping for circuit JPEG_X3 and TATE_X3.

TATE_X3 have 4.6 and 2.9 times more grids than JPEG and TATE respectively, the inference time increases linearly in proportion to the number of grids. Since the prediction stage can be performed quickly by using GPUs, DEF parsing and feature extraction stage account for the majority of the inference time, and the execution time of these two stages is linearly related to the number of grids. It is because the number of input parameters to be extracted is proportional to the number of grids. Also, for the large-size circuits, the inference time is within 10 minutes which is extremely shorter than the detailed routing time (about 6 hours).

Fig 3.15 shows the trend of F1-score according to the window size and sliding stride when performing input cropping for circuit JPEG_X3 and TATE_X3. The overlap ratio in the y-axis of the figure indicates what percentage of the two adjacent windows overlap with the size of the window. For example, if the window size is 400×400 and overlap ratio is 10%, two adjacent windows share 40 grids in between them, and sliding stride becomes 360×360.

Experimental results in Fig 3.15 demonstrate that both JPEG_X3 and TATE_X3 show F1-score improvement as the window size increases. It is because a larger window size enables the prediction model to consider the huge amount of neighboring information effectively. However, for the overlap ratio, the two circuits show different

Table 3.10: Statistics for the macro circuits from the ISPD 2015 benchmarks used in experiments.

Design	#Macro	#Standard cells	#Placements
MGC_EDIT_DIST_A	6	97,099	94
MGC_FFT_A	6	35,449	94
MGC_DES_PERF_A	4	136,059	60
MGC_MATRIX_MULT_A	5	196,061	14
MGC_MATRIX_MULT_B	7	175,813	15

trends. In circuit TATE_X3, overlap ratio changes do not impact the F1-score significantly. On the other hand, in the case of JPEG_X3, F1-score is the highest when the overlap ratio is 10%, where the F1-score decreases as the overlap ratio increase further. In common, both circuits show the highest F1 when the window size is 400 and the overlap ratio is 5$\sim$10%.

• **Adaptation to macro designs**: Macros are an essential block in SoC designs, and it is known that the presence of macro greatly affects the distribution of DRV [55]. Since the opencore circuits used in the experiment of section 3.3.3 do not have macro blocks, it is difficult to measure the capability of PGNN for the SoC design. We synthesized five ISPD 2015 benchmarks [84] under Nangate 15nm cell library using Synopsys Design Compiler and IC Compiler 2. Detailed statistics of the designs are shown in Table 3.10. We generated a total of 277 data of macro circuits by varying macro floorplanning and input placement parameters. To consider the effect of macros on DRV, we additionally extracted macro density which means the ratio of the area occupied by the macros inside the grid.

Table 3.11 shows a performance comparison of RouteNet [55] and PGNN at different training and test set coverage. ”Macro” in the training and test set column indicates the circuits in Table 3.10 and ”Opencore” represents the circuits in Table 3.1. 20% of each circuit data is randomly selected as a test set, and the other data are classi-

Table 3.11: Comparison of the performance of DRC hotspot prediction, in terms of F1-score, of RouteNet and PGNN for macro and opencore circuits.

Training set	Test set	Model	Precision	Recall	FPR	F1
Macro	Macro	RouteNet	74.62%	71.67%	0.37%	73.11%
		PGNN	78.37%	75.73%	0.31%	**77.03%**
Opencore +Macro	Macro	RouteNet	72.70%	69.63%	0.39%	71.13%
		PGNN	78.45%	72.83%	0.30%	**75.54%**
Opencore	Opencore	RouteNet	69.70%	70.12%	0.82%	69.91%
		PGNN	74.35%	74.53%	0.69%	**74.44%**
Opencore +Macro	Opencore	RouteNet	69.42%	69.37%	0.82%	69.39%
		PGNN	74.02%	75.00%	0.70%	**74.51%**

fied as a training set. Experimental results show that PGNN achieves about 4% higher F1-score on the macro test set than RouteNet, which is known to show high prediction performance on macro design. Interestingly, when the opencore design is added to the training set, the prediction performance for macro design is downgraded on both models. It seems that DRVs usually occur at the boundary of macros in SoC design, and the prediction of this trend might be weakened as the training set includes the opencore circuits without macros. On the other hand, PGNN outperforms RouteNet on opencore test set, achieving about 5% high F1-score. This is because PGNN effectively considers the pin accessibility of the standard cells by utilizing pin proximity graph as an input.

3.3.5 Handling Data Imbalance Problem in Regression Model

To implement efficient placement improvement methodology, predicting how many DRVs occur in the DRC hotspot grids is an important problem so that placement improvement engine is able to focus on optimizing grids expected to occur a lot of DRVs.

To efficiently predict the number of DRVs in each grid, which is known as a regression problem in machine learning domain, our regression model denoted as PGNN_REG removes the sigmoid layer at the final stage of PGNN and is trained towards the number of DRVs with weighted L2 MSE (Mean-Square-Error) loss function, which is formulated as follows:

$$Loss(y_N, \hat{y}) = \sum_{N=0} w_N (y_N - \hat{y})^2 \tag{3.5}$$

where w_N is the weight assigned to class N which is composed of grids that have N DRVs. To alleviate the data imbalance problem, we assign a higher weight to the minority classes so that the loss function highly focuses on the accurate prediction of these classes. To find the best weight assignment, we compare the following three weight assignment strategies.

- **Uniform weighting** : $w_N = 1$ for all classes.
- **Positive class weighting** : $w_0 = 1$ for class 0, $w_N = \frac{\#grids\ of\ class\ 0}{\#grids\ of\ other\ classes}$ for other classes.
- **Classwise weighting** : $w_N = \frac{\#grids\ of\ class\ 0}{\#grids\ of\ other\ classes}$ for all classes.

Table 3.12 shows the prediction accuracy and average error of each class for the three weighting strategies. Note that class 0 accounts for 97.7% of total data and the other classes each have less than 1% of total data, which shows extremely imbalanced data distribution. The prediction result is rounded for classification. Accuracy of class N denotes the ratio of grids predicted accurately to class N by the total number of grids in class N, The accuracy of class N denotes the number of grids predicted accurately to class N by the total number of grids in class N, and Avg. error of class N indicates the average L1 loss of the grids in class N.

Experimental results show that all three strategies commonly show low accuracy and high average error for the high DRV classes. The uniform weighting strategy has the best accuracy and average error in class 0, but the worst performance in the other

Table 3.12: Comparison of prediction accuracy and average prediction error between the three weighting strategies for individual output classes.

#DRV in grid		**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**$\geq$ 10**
Percentage		97.7%	0.94%	0.48%	0.29%	0.18%	0.12%	0.09%	0.06%	0.04%	0.03%	0.02%
Uniform weighting	Accuracy	**98.8%**	40.6%	22.0%	14.3%	10.4%	9.1%	5.4%	5.1%	8.8%	2.5%	0.0%
	Avg. Error	**0.021**	0.635	1.087	1.516	1.954	2.347	2.862	3.320	3.629	4.155	5.005
Positive class weighting	Accuracy	93.5%	**59.8%**	**38.0%**	10.8%	6.8%	3.5%	2.2%	1.4%	0.7%	1.2%	0.0%
	Avg. Error	0.160	**0.484**	**0.726**	1.311	1.949	2.562	3.323	3.872	4.433	5.107	6.014
Classwise weighting	Accuracy	75.3%	16.0%	30.3%	**28.4%**	**21.6%**	**20.0%**	**18.7%**	**15.9%**	**11.0%**	**4.9%**	**2.8%**
	Avg. Error	0.581	1.412	1.199	**1.184**	**1.310**	**1.444**	**1.574**	**1.867**	**2.130**	**2.486**	**3.139**

classes, which does not properly handle the data imbalance problem. The positive class weighting strategy shows superior performance in classes 1 and 2, and it seems this strategy is not effective over class 2 which has less number of grids compared to classes 1 and 2, because the same weight is applied for all classes except class 0. On the other hand, classwise weighting more focuses on the classes with high DRVs, outperforming the other strategies over class 2.

Fig 3.16 illustrates prediction distribution comparison of three weighting methodologies for benchmark NOVA. Prediction results for each class are displayed by a box, where the orange line is a median, the upper and lower edges of the boxes indicate 25th and 75th percentile, the straight line outside the box is the maximum and minimum data point excluding outliers, and circle points illustrate outliers. The dotted line from left-bottom to right-top indicates perfect prediction. In general, as the ground-truth increases, the predicted value increases accordingly but shows lower prediction results than the ground-truth in a high DRV range. When comparing Fig 3.16 (a) and (b), positive class weighting does not significantly change the prediction range. However, classwise weighting (see Fig 3.16 (c)) significantly increases the prediction results for all classes, and in the high DRV region, its results are closer to the dotted line than the other two methods, but the predictions of the low DRV region are overestimated.

Table 3.13 shows the coefficient of determination R^2 comparison of the three weighting strategies for all benchmarks. Large R^2 indicates a high correlation between ground truth and prediction. Experimental results show that the uniform weighting strategy outperforms other methods in all benchmarks, apparently because it is relatively accurate in predicting class 0, which overwhelms the other classes in terms of the number of grids. However, since the accurate prediction of high ground-truth classes is important for early stage placement improvement, uniform and classwise weighting can be used selectively depending on the user's purpose.

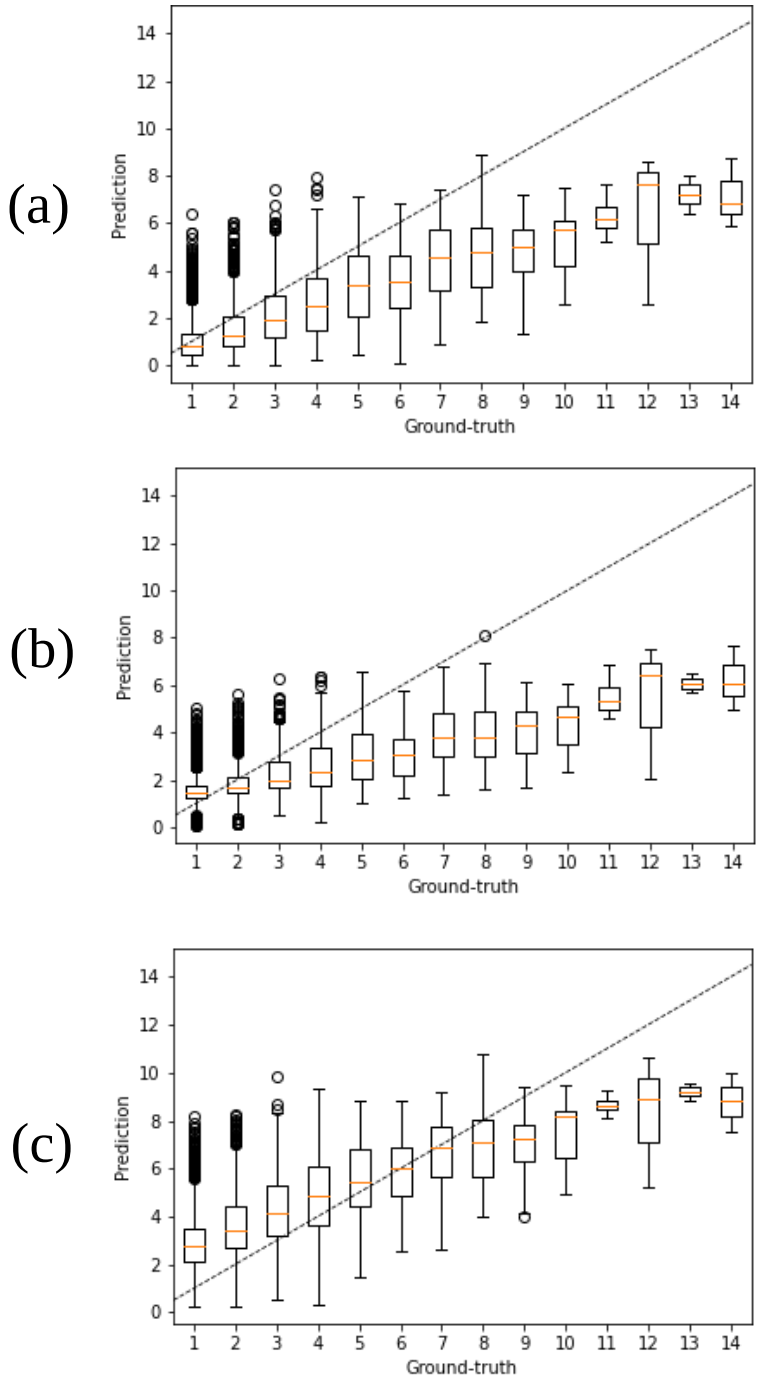


Figure 3.16: Visualization of DRV regression results for benchmark NOVA. (a) uniform weighting (b) positive class weighting (c) classwise weighting.

Table 3.13: R^2 comparison between the three weighting strategies for all benchmarks.

Benchmark	**Uniform weighting**	**Positive class weighting**	**Classwise weighting**
AES	**0.497**	0.295	0.127
B18	**0.228**	0.111	0.059
B19	**0.269**	0.142	0.080
ECG	**0.553**	0.428	0.289
ETH	**0.527**	0.402	0.233
JPEG	**0.569**	0.401	0.255
NOVA	**0.626**	0.457	0.375
TATE	**0.603**	0.403	0.216
VGA	**0.514**	0.357	0.289
Avg.	**0.487**	0.333	0.214

Chapter 4

Conclusions

4.1 Chapter 2

In chapter 2, we proposed an integrated approach to the two problems of transistor folding and placement. Precisely, we proposed a *globally optimal* algorithm of search tree based design space exploration, devising a set of four effective speeding up techniques as well as dynamic programming based fast cost computation. Our algorithm also incorporated the minimum oxide diffusion jog constraint. In addition, to make an effective cell layout synthesis flow down to in-cell routing, we provided a fast in-cell routability estimation metric to be used in transistor placement and a method to explore cell layouts by varying the cell size constraint. Through experiment with all transistor netlists and design rules in the ASAP 7nm cell library, it was shown that our proposed method was able to synthesize fully routable cell layouts of minimal size within a very fast time in the ASAP 7nm library, outperforming the cell layout quality in the ASAP 7nm library, which otherwise, might take several hours or days to manually complete layouts of the quality level comparable to ours.

4.2 Chapter 3

In chapter 3, we addressed the problem of DRC hotspot prediction at the placement stage. In comparison with the conventional ML (machine learning) based models, which invariably revealed ineffectiveness on assembling the aggregate data on (1) pin accessibility and (2) routing congestion, this work proposed a novel ML based DRC hotspot prediction model called PGNN, which was able to accurately capture the combined impact of items 1 and 2 on DRC hotspots by devising a new graph representation so-called *pin proximity graph* and developing a tightly combined ML model of GNN and U-net. In the meantime, through experiments using Nangate 15nm library, it was confirmed that our PGNN consistently outperformed the existing ML models, achieving on average 7.8$\sim$12.5% improvements on F1-score while taking 5.5$\times$ fast inference time over the existing state-of-the-art technique. In the future, we want to develop a placement optimization methodology which installs PGNN as a core engine.

Bibliography

[1] V. Vashishtha, M. Vangala, and L. T. Clark, "Asap7 predictive design kit development and cell design technology co-optimization," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 992–998.

[2] K. Jo and T. Kim, "Optimal transistor placement combined with global in-cell routing in standard cell layout synthesis," in *2021 IEEE 39th International Conference on Computer Design (ICCD)*. IEEE, 2021, pp. 517–524.

[3] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[4] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[5] P. Spindler and F. M. Johannes, "Fast and accurate routing demand estimation for efficient routability-driven placement," in *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2007, pp. 1–6.

[6] S. Chakravarty, X. He, and S. Ravi, "Minimum area layout of series-parallel transistor networks is np-hard," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 7, pp. 943–949, 1991.

[7] A. Gupta and J. P. Hayes, "Optimal 2-d cell layout with integrated transistor folding," in *Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, 1998, pp. 128–135.

[8] A. Lu, H.-J. Lu, E.-J. Jang, Y.-P. Lin, C.-H. Hung, C.-C. Chuang, and R.-B. Lin, "Simultaneous transistor pairing and placement for cmos standard cells," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015, pp. 1647–1652.

[9] C.-Y. Hwang, Y.-C. Hsieh, Y.-L. Lin, and Y.-C. Hsu, "A fast transistor-chaining algorithm for cmos cell layout," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 7, pp. 781–786, 1990.

[10] J. Cortadella, "Area-optimal transistor folding for 1-d gridded cell design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 11, pp. 1708–1721, 2013.

[11] 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)*. IEEE, 2018, pp. 500–506.

[12] K. Jo, S. Ahn, J. Do, T. Song, T. Kim, and K. Choi, "Design rule evaluation framework using automatic cell layout generator for design technology co-optimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1933–1946, 2019.

[13] T. Iizuka, M. Ikeda, and K. Asada, "Exact minimum-width multi-row transistor placement for dual and non-dual cmos cells," in *2006 IEEE International Symposium on Circuits and Systems*. IEEE, 2006, pp. 4–pp.

[14] A. Sorokin and N. Ryzhenko, "Sat-based placement adjustment of finfets inside unroutable standard cells targeting feasible drc-clean routing," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 2019, pp. 159–164.

[15] D. Lee, D. Park, C.-T. Ho, I. Kang, H. Kim, S. Gao, B. Lin, and C.-K. Cheng, "Sp&r: Smt-based simultaneous place-&-route for standard cell synthesis of advanced nodes," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[16] M. Guruswamy, R. L. Maziasz, D. Dulitz, S. Raman, V. Chiluvuri, A. Fernandez, and L. G. Jones, "Cellerity: A fully automatic layout synthesis system for standard cell libraries," in *Proceedings of the 34th annual Design Automation Conference*, 1997, pp. 327–332.

[17] Y.-L. Li, S.-T. Lin, S. Nishizawa, H.-Y. Su, M.-J. Fong, O. Chen, and H. Onodera, "Nctucell: A dda-aware cell library generator for finfet structure with implicitly adjustable grid map," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[18] T.-C. Lee, C.-Y. Yang, and Y.-L. Li, "itplace: machine learning-based delay-aware transistor placement for standard cell synthesis," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–8.

[19] H. Ren and M. Fojtik, "Nvcell: Standard cell layout in advanced technology nodes with reinforcement learning," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 1291–1294.

[20] P. Cremer, S. Hougardy, J. Schneider, and J. Silvanus, "Automatic cell layout in the 7nm era," in *Proceedings of the 2017 ACM on International Symposium on Physical Design*, 2017, pp. 99–106.

[21] P. Van Cleeff, S. Hougardy, J. Silvanus, and T. Werner, "Bonncell: Automatic cell layout in the 7-nm era," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2872–2885, 2019.

[22] N. Ryzhenko and S. Burns, “Standard cell routing via boolean satisfiability,” in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 603–612.

[23] J. Cortadella, J. Petit, S. Gomez, and F. Moll, “A boolean rule-based approach for manufacturability-aware cell routing,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 3, pp. 409–422, 2014.

[24] W. Ye, B. Yu, D. Z. Pan, Y.-C. Ban, and L. Liebmann, “Standard cell layout regularity and pin access optimization considering middle-of-line,” in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, 2015, pp. 289–294.

[25] N. Ryzhenko, S. Burns, A. Sorokin, and M. Talalay, “Pin access-driven design rule clean and dfm optimized routing of standard cells under boolean constraints,” in *Proceedings of the 2019 International Symposium on Physical Design*, 2019, pp. 41–47.

[26] Y.-L. Li, S.-T. Lin, S. Nishizawa, and H. Onodera, “Mcell: multi-row cell layout synthesis with resource constrained max-sat based detailed routing,” in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–8.

[27] H. Ren and M. Fojtik, “Standard cell routing with reinforcement learning and genetic algorithm in advanced technology nodes,” in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021, pp. 684–689.

[28] S. Chung, J. Jeong, and T. Kim, “Improving performance and power by co-optimizing middle-of-line routing, pin pattern generation, and contact over active gates in standard cell layout synthesis,” in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2022, pp. 1–6.

[29] J. Kim and S.-M. Kang, "An efficient transistor folding algorithm for row-based cmos layout design," in *Proceedings of the 34th annual Design Automation Conference*, 1997, pp. 456–459.

[30] K. Han, A. B. Kahng, and H. Lee, "Scalable detailed placement legalization for complex sub-14nm constraints," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2015, pp. 867–873.

[31] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[32] S. Liu, Y. Pu, P. Liao, H. Wu, R. Zhang, Z. Chen, W. Lv, Y. Lin, and B. Yu, "Fastgr: Global routing on cpu-gpu with heterogeneous task graph scheduler," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[33] S. Lin, J. Liu, E. F. Young, and M. D. Wong, "Gamer: Gpu accelerated maze routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[34] K. Baek, H. Park, S. Kim, K. Choi, and T. Kim, "Pin accessibility and routing congestion aware drc hotspot prediction using graph neural network and u-net," in *2022 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2022, pp. 1–8.

[35] Y.-C. Lu, S. Nath, V. Khandelwal, and S. K. Lim, "Doomed run prediction in physical design by exploiting sequential flow and graph learning," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[36] V. A. Chhabria, Y. Zhang, H. Ren, B. Keller, B. Khailany, and S. S. Sapatnekar, "Mavirec: Ml-aided vectored ir-drop estimation and classification," in *2021 De-*

sign, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2021, pp. 1825–1828.

[37] Y. Zhang, H. Ren, and B. Khailany, "Grannite: Graph neural network inference for transferable power estimation," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[38] A. Hosny, S. Hashemi, M. Shalan, and S. Reda, "Drills: Deep reinforcement learning for logic synthesis," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020, pp. 581–586.

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

[40] H. Liao, W. Zhang, X. Dong, B. Poczos, K. Shimada, and L. Burak Kara, "A deep reinforcement learning approach for global routing," *Journal of Mechanical Design*, vol. 142, no. 6, 2020.

[41] 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," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 84003. American Society of Mechanical Engineers, 2020, p. V11AT11A002.

[42] Y.-C. Lu, S. Nath, V. Khandelwal, and S. K. Lim, "Rl-sizer: Vlsi gate sizing for timing optimization using deep reinforcement learning," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 733–738.

[43] A. Agnesina, K. Chang, and S. K. Lim, "Vlsi placement parameter optimization using deep reinforcement learning," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[44] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[45] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[46] J. Seo, J. Jung, S. Kim, and Y. Shin, "Pin accessibility-driven cell layout redesign and placement optimization," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2017, pp. 1–6.

[47] Y. Ding, C. Chu, and W.-K. Mak, "Pin accessibility-driven detailed placement refinement," in *Proceedings of the 2017 ACM on International Symposium on Physical Design*, 2017, pp. 133–140.

[48] S. Kim, K. Jo, and T. Kim, "Boosting pin accessibility through cell layout topology diversification," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021, pp. 183–188.

[49] S. Kim and T. Kim, "Pin accessibility-driven placement optimization with accurate and comprehensive prediction model," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022.

[50] W.-T. J. Chan, Y. Du, A. B. Kahng, S. Nath, and K. Samadi, "Beol stack-aware routability prediction from placement using data mining techniques," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*. IEEE, 2016, pp. 41–48.

[51] W.-T. J. Chan, P.-H. Ho, A. B. Kahng, and P. Saxena, "Routability optimization for industrial designs at sub-14nm process nodes using machine learning," in *Proceedings of the 2017 ACM on International Symposium on Physical Design*, 2017, pp. 15–21.

[52] W. Zeng, A. Davoodi, and R. O. Topaloglu, "Explainable drc hotspot prediction with random forest and shap tree explainer," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 1151–1156.

[53] W.-T. Hung, J.-Y. Huang, Y.-C. Chou, C.-H. Tsai, and M. Chao, "Transforming global routing report into drc violation map with convolutional neural network," in *Proceedings of the 2020 International Symposium on Physical Design*, 2020, pp. 57–64.

[54] C. Yu and Z. Zhang, "Painting on placement: Forecasting routing congestion using conditional generative adversarial nets," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[55] Z. Xie, Y.-H. Huang, G.-Q. Fang, H. Ren, S.-Y. Fang, Y. Chen, and J. Hu, "Routenet: Routability prediction for mixed-size designs using convolutional neural network," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[56] T.-C. Yu, S.-Y. Fang, H.-S. Chiu, K.-S. Hu, P. H.-Y. Tai, C. C.-F. Shen, and H. Sheng, "Pin accessibility prediction and optimization with deep learning-based pin pattern recognition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[57] ——, "Lookahead placement optimization with cell library-based pin accessibility prediction via active learning," in *Proceedings of the 2020 International Symposium on Physical Design*, 2020, pp. 65–72.

[58] R. Liang, H. Xiang, D. Pandey, L. Reddy, S. Ramji, G.-J. Nam, and J. Hu, "Drc hotspot prediction at sub-10nm process nodes using customized convolutional network," in *Proceedings of the 2020 International Symposium on Physical Design*, 2020, pp. 135–142.

[59] K. Baek and T. Kim, "Simultaneous transistor folding and placement in standard cell layout synthesis," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–8.

[60] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical society*, vol. 7, no. 1, pp. 48–50, 1956.

[61] L. d. Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[62] N. Bjørner, A.-D. Phan, and L. Fleckenstein, "νz-an optimizing smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2015, pp. 194–199.

[63] H. Wang, K. Wang, J. Yang, L. Shen, N. Sun, H.-S. Lee, and S. Han, "Gcn-rl circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[64] H. Ren, G. F. Kokai, W. J. Turner, and T.-S. Ku, "Paragraph: Layout parasitics and device parameter prediction using graph neural networks," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[65] Y.-C. Lu, S. S. K. Pentapati, L. Zhu, K. Samadi, and S. K. Lim, "Tp-gnn: A graph neural network framework for tier partitioning in monolithic 3d ics," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[66] R. Liang, Z. Xie, J. Jung, V. Chauha, Y. Chen, J. Hu, H. Xiang, and G.-J. Nam, "Routing-free crosstalk prediction," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[67] Y. Li, Y. Lin, M. Madhusudan, A. Sharma, W. Xu, S. S. Sapatnekar, R. Harjani, and J. Hu, "A customized graph neural network model for guiding analog ic placement," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[68] W. Li, Y. Ma, Y. Lin, and B. Yu, "Adaptive layout decomposition with graph embedding neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[69] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[70] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[71] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[72] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[73] C. Chu, "Flute: Fast lookup table based wirelength estimation technique," in *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.* IEEE, 2004, pp. 696–701.

[74] M. Pan and C. Chu, "Fastroute: A step to integrate global routing into placement," in *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, 2006, pp. 464–471.

[75] J. Chen, J. Kuang, G. Zhao, D. J.-H. Huang, and E. F. Young, "Pros: A plug-in for routability optimization applied in the state-of-the-art commercial eda tool

using deep learning," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–8.

[76] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[77] Y. Wu and K. He, "Group normalization," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19.

[78] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[79] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[80] Oliscience, "Opencores," 1999. [Online]. Available: https://opencores.org

[81] M. Martins, J. M. Matos, R. P. Ribas, A. Reis, G. Schlinker, L. Rech, and J. Michelsen, "Open cell library in 15nm freepdk technology," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, 2015, pp. 171–178.

[82] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[83] ——, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[84] I. S. Bustany, D. Chinnery, J. R. Shinnerl, and V. Yutsis, "Ispd 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, 2015, pp. 157–164.

초 록

칩 구현의 물리적 설계 단계에서, 높은 성능의 표준 셀 설계와 배선 연결 이후 조기에 설계 규칙 위반을 정확히 예측하는 것은 최신 공정에서 특히 중요한 문제이다. 본 논문에서는 물리적 설계에서의 설계 품질과 총 설계 시간 향상을 달성할 수 있는 두 가지 방법론을 제안한다.

먼저, 본 논문에서는 표준 셀 레이아웃 합성에서 트랜지스터 폴딩과 배치를 종합적으로 진행할 수 있는 방법론을 논한다. 구체적으로 탐색 트리 기반의 최적화 알고리즘과 동적 프로그래밍 기반 빠른 비용 계산 방법과 여러 속도 개선 기법을 제안한다. 여기에 더해, 최신 공정에서 트랜지스터 폴딩과 배치로 인해 발생할 수 있는 최소 산화물 확산 영역 설계 규칙을 고려하였다. 최신 공정에 대한 표준 셀 합성 실험 결과, 본 논문에서 제안한 방법이 설계 전문가가 수동으로 설계한 것 대비 높은 성능을 보이고, 설계 시간도 매우 짧음을 보인다.

두번째로, 본 논문에서는 셀 배치 단계에서 핀 접근성과 연결 혼잡으로 인한 영향을 종합적으로 고려할 수 있는 머신 러닝 기반 설계 규칙 위반 구역 예측 방법론을 제안한다. 먼저 표준 셀의 입/출력 핀의 물리적 정보와 핀과 핀 사이 방해 관계를 효과적으로 표현할 수 있는 핀 근접 그래프를 제안하고, 그래프 신경망과 유넷 신경망을 효과적으로 결합한 새로운 형태의 머신 러닝 모델을 제안한다. 이 모델에서 그래프 신경망은 핀 근접 그래프로부터 핀 접근성 정보를 추출하고, 유넷 신경망은 격자 기반 특징으로부터 연결 혼잡 정보를 추출한다. 실험 결과 본 논문에서 제안한 방법은 이전 연구들 대비 더 빠른 예측 시간에 더 높은 예측 성능을 달성함을 보인다.

주요어: 표준셀, 트랜지스터 배치, 트랜지스터 폴딩, 설계 규칙 위반, 핀 접근성
학번: 2017-21739