MACHINE-LEARNING TECHNIQUES FOR VLSI DESIGN AUTOMATION

A Dissertation

by

RONGJIAN LIANG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Jiang Hu
Committee Members,	Weiping Shi
	Duncan M. Walker
	Anxiao Jiang
Head of Department,	Scott Schaefer

December 2021

Major Subject: Computer Engineering

ABSTRACT

VLSI design productivity has already become the bottleneck to take full advantage of potential benefit brought by technology scaling, leading to the famous design productivity gap, i.e., the mismatch between the available transistor density and the transistor density that designers can handle. Advancement of VLSI design automation, or called EDA, is in urgent need to narrow down the gap.

The strong learning capability of modern machine learning (ML) techniques make them a good fit to cope with the high complexity in EDA tasks via discovering knowledge from past experience and applying them to new designs. However, it is extremely time consuming to obtain new training data from chip designs and existing data is usually noisy and disorganized. Thus, a main challenge faced by ML EDA is how to develop effective solutions with the poor quality data. Rather than plugin usage of existing ML techniques, it is necessary to customize ML methodologies for EDA tasks, mainly from three aspects: data, algorithm and domain knowledge, guided by insights into problem characteristics and understanding of ML methodologies.

We explore the potential of ML techniques in four representative EDA tasks as follows:

- Design Rule Check (DRC) Hotspot Prediction: Design Rule Violation (DRV) prediction at early layout stages is critical to achieve freedom from DRV. We target at DRC hotspot prediction, i.e., classifying layout regions that are subject to DRVs, at cell placement stage without depending on global routing information. A customized convolutional network architecture is proposed to handle mixed resolution input and output.

- DRC Heatmap Prediction: In this task we target at the DRV density regression prediction problem, which provides more detailed information in guiding DRV mitigation techniques than DRC hotspot prediction. Besides, the challenge of noisily labeled data caused by non-deterministic parallel routing is addressed by combining the strong modeling capability of deep neural networks and the strength of stochastic models in coping with uncertainty.

- Routing-Free Crosstalk Prediction: Increasingly-smaller interconnect spacing in advanced technology nodes makes crosstalk a significant threat to signal integrity and timing. Given a placement, we identify physical, electrical and logical features that affect crosstalk-induced noise and delay. We then employ ML techniques to train the crosstalk prediction models, which can be used to identify crosstalk-critical nets at placement stages.

- EDA Flow Parameter Tuning: We develop an automatic flow tuning tool for efficient parameter tuning of VLSI design flows. It utilizes both *exploitation* using transferred parameter knowledge from archived data from legacy designs and *exploration* via a multi-stage cooperative co-evolutionary framework. Furthermore, novel flow jump-start and early-stop techniques are developed to reduce the overall runtime for tuning. In addition, a concurrent multi-trade-off learning technique is proposed to enable truly multi-objective tuning by facilitating efficient Pareto front exploration.

The above are examples of customizing ML techniques for EDA problems and promising results have been achieved. We hope this dissertation can be instructive for further exploration in the direction of applying ML to EDA tasks.

DEDICATION

To my dear parents.

ACKNOWLEDGMENTS

I pay my deep gratitude to my advisor, Dr. Jiang Hu, for his patient guidance and insightful advises during my Ph.D. study. I learnt not only technical skills but also general problem solving strategies from him. I am also deeply impressed by his passion for research. My entire life will be benefited from the experience I got in my Ph.D. study. I would like to thank Gi-Joon Nam from IBM research, for sharing his industrial insights. He is always willing to discuss with us whenever we have difficulties.

I would like to thank my committee members, Prof. Weiping Shi of the Department of Electrical & Computer Engineering, Prof. Duncan M. “Hank" Walker and Prof. Anxiao Jiang of the Department of Computer Science & Engineering, for their helpful suggestions and comments on my research and dissertation. Thanks to my mates in Texas A&M University, Wenbin Xu, Hongxin Kong, Lin Huang, Yanxiang Yang, Jianfeng Song, Hailiang Hu, Yaguang Li, Erick Carvajal Barboza, Nithyashankari Gummidipoondi Jayasan, Yishuang Lin and Donghao Fang. Also, I would like to thank Hua Xiang, Jinwook Jung, Diwesh Pandey, Lakshmi Reddy, Shyam Ramji Shyam Ramji from IBM research for their dependable supports. Finally, I would like to thank my parents for their support during my Ph.D. study.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Prof. Jiang Hu and Prof. Weiping Shi of the Department of Electrical & Computer Engineering and Prof. Duncan M. “Hank" Walker and Prof. Anxiao Jiang of the Department of Computer Science & Engineering.

All the work conducted for the dissertation was completed by the student independently.

Funding Sources

This graduate study was partially supported by Semiconductor Research Corporation Tasks 2810.021 and 2810.022 through UT Dallas’ Texas Analog Center of Excellence (TxACE).

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

1.1 Challenges Faced by Conventional VLSI Design Automation Flows

Very-Large-Scale-Integration (VLSI) circuits have a wide range of applications, such as microprocessors in personal computers, chips in smart phones and embedded processors in robotics. As the feature size of semiconductor process technology keeps scaling down, modern VLSI technology can integrate more than billions of transistors on one chip [9]. On the one hand, it endows VLSI circuits with ever-growing computation power; on the other hand, it imposes great challenges to VLSI design. The enormous circuit size, growing process variation and stricter design margins coming along with advanced technology nodes make VLSI design flow dramatically complicated, usually involving hundreds of thousands of steps [10]. Besides, since numerous problems in VLSI design flows are large scale NP-hard problems, a plethora of heuristic algorithms with unknown suboptimality have been developed, which lead to high unpredictability in design flows. VLSI design productivity has already become the bottleneck to take full advantage of potential benefit brought by technology scaling. As highlighted by the 2013 ITRS roadmap [11], the design productivity gap: mismatch between the available transistor density and the transistor density that designers can handle, has become widened as semiconductor process nodes keep shrinking (as plotted in Figure 1.1). Advancement of VLSI design automation, or called Electronic Design Automation (EDA), is in urgent need to narrow down the design productivity gap.

As summarized in [1], the design productivity gap mainly reflects in three aspects of VLSI design, i.e., cost, quality and predictability. Cost includes computation effort, engineering effort, etc. VLSI design is computationally expensive. Figure 1.2 shows a typical VLSI design flow, which involves multiple design stages and many forward-and-feedback loops. It is not uncommon to take several days to go through the RTL-to-GDSII process for a modern VLSI circuit. What is worse, as shown in Figure 1.2, the forward-and-feedback loops typically expect “expert feedback" from designers, and also, designers usually need to spend significant amount of effort on analyzing and

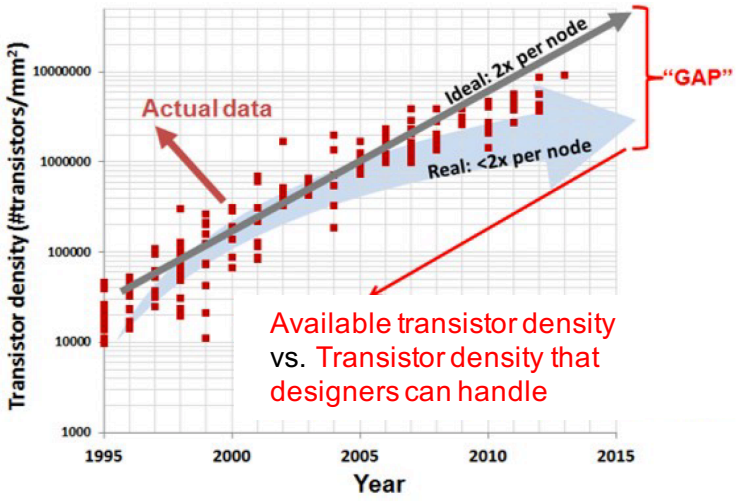


Figure 1.1: Design productivity gap (adapted from [1]).

fixing remaining problems manually after the tool flow is completed. With the growing complexity and increasing turn-around-time (TAT) of each design step, the demand for design expertise soars while the supply grows slowly. Shortage of design expertise seriously widens the design productivity gap. Quality corresponds to power, performance and area (PPA) metrics, along with other requirements such as signal integrity and freedom from Design Rule Violations (DRVs). At advanced technology nodes, the strict design margins, complicated design rules and limited layout resources make the optimization of quality of results (QoR) extremely difficult. Predictability corresponds to the reliability of the design schedule. For example, design schedules are affected by whether there will be more than expected place&route iterations to achieve timing closure or whether a given placement solution is routable. Also, the outcome QoR should be predictable. However, as more and more complicated heuristic algorithms are involved, the reliability and predictability of conventional EDA flows keep decreasing.

1.2 Opportunities of Machine Learning for VLSI Design Automation

Machine learning (ML) has seen flourishing development and a wide range of applications in recent years [12]. It is a natural idea to take advantage of developed theories and techniques in the ML community and reproduce the success in the EDA field as like in other applications.

VLSI Design Automation problems usually have large scale, strong nonlinearities, and high-

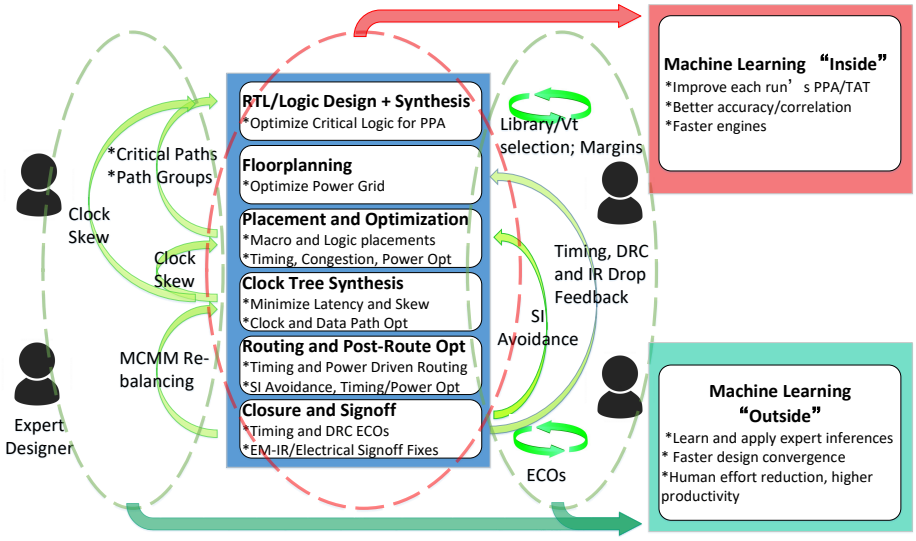


Figure 1.2: Opportunities of ML for EDA (adapted from [2]).

order interactions among circuit modules and design stages. This level of complexity makes it difficult to explicitly encode their solutions into programs. The strong learning capability of modern ML techniques make them a good fit to cope with such complexity via discovering knowledge from past experience and apply them to new designs.

Cadence [13], a leading EDA company, categorizes applications of ML to the EDA field into two classes: ML inside (flow) and ML outside (flow). While ML inside targets improving the results from one part of the flow, ML outside is about orchestrating multiple design stages by learning and applying expert inferences. Both can create great benefits to VLSI design in terms of cost, quality and predictability. Examples of ML inside include but not limit to:

- Predicting downstream effects at an early stage to promote predictable design flow;
- Taking proactive actions based on prediction results to improve PPA and eliminate unnecessary forward-and-backward loops;
- Providing better correlation between PPA analysis results of different tools/models to promote fast and cheap analysis;

ML outside is about flow-level learning and typically orchestrates a couple of stages by learning and applying expert inference. Examples include but not limit to:

- Automatic EDA tool parameter (knobs) tuning, constraints and design margins determination at each design for better PPA and reduced cost;
- Smart flow management that can perform what-if's, trace back and efficient resource allocation.

1.3 Previous Works in Machine Learning for VLSI Design Automation and New Challenges

There have been some successful ML applications to VLSI design automation, as reviewed in [14]. A majority of them belong to the ML inside category, targeting at different design stages such as system-level design space exploration and high-level synthesis [15–25], logic synthesis [26–30], physical design [31–45], verification and testing [46–54].

System-level design space exploration optimizes architecture-level parameters, such as the 3D network-on-chip [15] and configurations of processor core [16,17,21]. High-level synthesis schedules operations and allocates them to functional units. Surrogate models are developed to obtain fast estimations of PPA for given high-level synthesis solutions [18–20, 22, 25]. A deep reinforcement learning-based scheduler for FPGA is implemented in [23] to directly optimize scheduling solutions. An ML-based method is proposed in [24] to set meta-heuristic specific parameters for high-level synthesis design space exploration.

Logic synthesis optimizes the transformation from the register-transfer-level representation of a design to a gate-level description at the target technology node. Various types of ML models are deployed as surrogate models to obtain fast estimation of properties of the design, e.g., parasitic [27], power [28] and error rates [29]. A reinforcement learning-based engine is developed in [30] to directly perform logic synthesis.

Physical design transforms the gate-level description of a design to an optimized geometrical representation. A typical physical design flow involves multiple stages, i.e., floor planning and power network synthesis, placement, clock tree synthesis and routing. Reinforcement learning techniques are leveraged to address the placement of macros [55], the optimization of clock tree synthesis [32] and the order of nets to be routed [33]. Various types of ML models, e.g., convo-

lutional neural networks, graph neural networks and ensemble tree-based models, are utilized for the prediction of routability/DRV [34–38], IR drops [39–41], datapath related patterns [42] and post-routing timing [44] (at pre-routing stages).

Verification ensures that the synthesized design will perform correctly adhering to its specifications when manufactured; while testing verifies that the manufactured physical device has no manufacturing defect. Verification is usually conducted via simulations. To speed up simulations, ML techniques are used to model DRAM access latency [46] and power waveforms of hardware accelerators [47]. An unsupervised learning-based method is developed in [48] to increase test coverage. Anomaly detection techniques are utilized in [49, 50] to detect manufacturing defects.

A smaller set of research has been done on ML outside [56–59]. In [56] a budget of tool runs is intelligently scheduled across different synthesis target frequencies to achieve high post-routing frequence subject to constraints on area and total power. ML-aided EDA flow parameter tuners are developed in [57, 58], automatically determining flow knobs for high QoR. In [59], an ML-based routing congestion prediction model is integrated into a placement engine for explicitly guiding the cell movement.

This dissertation focuses on applying ML to VLSI logic synthesis and physical design stages, which suffer from expensive runtime and expect significant amount of expert efforts. Though recent years have seen some successful applications of ML to EDA, there are still many challenges to be solved. In particular, some outstanding challenges come from the following characteristics of data in the EDA field, which are still under-explored.

- **Small volume of data**: A large volume of high quality data is essential for making ML approaches work well. EDA problems are typically in the “small data" context, compared to other domains where ML methods have achieved great success, since designs, tools, cell libraries, manufacturing processes, and analysis methodologies are always updating in the EDA filed. Also, due to the expensive runtime of steps in the EDA flows, it is not easy to collect large data sets.

- **Noisy data**: Tools and algorithms used for VLSI design usually do not show deterministic

and stable behaviours, mainly due to their heuristic nature and parallel implementations. To be more specific, different design outcomes might be produced for equivalent inputs, and a small perturbation in input could lead to substantially different outcomes. From the point view of ML, the data for EDA problems is with serious noise, which makes the learning in ML methods very difficult.

- **Various data structures**: EDA data comes in a wide variety of structures. And different sources of information in different structures need to be considered simultaneously in typical EDA tasks. For example, both netlist information, represented by a graph with hyperedges, and layout information, described in an image-like format, are critical for routability prediction. It presents a serious challenge for ML techniques to integrate data with various structures simultaneously.

1.4 Key Ideas and the Scope of the Research

We believe that customization is a key to successful ML applications for EDA. Rather than plugin usage of existing ML techniques, it is necessary to customize ML methodologies for EDA tasks, mainly from three aspects: data, algorithm and domain knowledge, with insights to special problem natures and deep understanding of ML techniques.

- **Preparation, augmentation and representation of data.** In data preparation, it is supposed to generate diverse yet realistic samples. Here “realistic" means that the knowledge learnt from the data should be able to generalize to instances encountered in real world chip design. Also, the ways to divide data into training set and testing set, is supposed to refer to usage scenarios in design practise. Data augmentation should also be conducted in a way that makes sense to EDA problem. For instance, while random rotation is a popular data augmentation technique for image processing, it is not applicable to layout “images" since horizontal and vertical layout resource and requirement are not identical. Data representation has a significant impact on the selection of ML algorithms and learning efficiency. For example, location of cells and macros after placement can be described as tabular data, suit-

able for decision tree-based methods, support-vector-machine, etc; this information can also be plotted to an image, suitable for deep neural network methods. It is not difficult to find that the above processing of data needs knowledge both in the EDA domain and in the ML field.

- **Adaption, development and integration of ML techniques.** ML techniques are powerful tools for EDA tasks. Given an EDA task, we might firstly analyze its characteristics and check if the problem can be mapped to a similar problem that has been well solved via ML methods. If yes, we can simply borrow developed tools from the ML community. If the problem has signification difference from conventional ML applications, customization of ML techniques to the given task, or a brand-new methodology is necessary to be proposed. Complicated tasks might demand the integration of multiple ML techniques, conventional EDA solutions and even methodologies from other domains. Techniques are tools after all. Systematic integration of techniques as needed, with insight of special problem properties, often delivers the best results.

- **Integration of domain knowledge with ML methodologies.** VLSI experts have accumulated a huge amount of domain knowledge, ranging from high-level ones such as what are the bottleneck problems in design flows, to low-level ones such as which features are important for a specific task. On the one hand, smartly “injecting" domain knowledge into ML algorithms, or combining existing EDA solutions with ML techniques, might produce visibly superior performance than algorithms without domain knowledge. On the other hand, experts’ knowledge could be sub-optimal, incomplete, does not generalize well on new designs. How to systematically integrate domain knowledge with the learning capability of ML methodologies is also an interesting topic.

We explore the potential of ML techniques in four EDA tasks, as summarized in Table 1.1. They cover prediction and optimization problems, as well as ML inside and outside flow applications. To achieve the best performance for the studied cases, we develop customized solutions with

insights to special problem natures and understanding of ML methodologies. A brief introduction to the studied cases is provided as follows.

- Design Rule Check (DRC) Hotspot Prediction: Design Rule Violation (DRV) prediction at early layout stages is critical to achieve freedom from DRV, which is a fundamental requirement for chip designs. We target at DRC hotspot prediction, i.e., classifying layout regions that are subject to DRVs, at cell placement stage without depending on global routing information. We propose a customized convolutional network architecture that can simultaneously take high resolution pin images and low resolution placement information as input features. This approach is evaluated on 12 industrial designs at 7nm process node. Experiment results show that our approach considerably outperforms the previous works and has fast feature extraction and inference time as no global routing information is required.

- DRC Heatmap Prediction: We target at the DRV density regression problem, called DRC heatmap prediction, at cell placement stage, which provides more detailed information in guiding DRV mitigation techniques than DRC hotspot prediction. Besides, the challenge of noisily labeled data caused by non-deterministic parallel routing is addressed by combining the strong modeling capability of deep neural network and the strength of stochastic models in coping with uncertainty. Experiments results on industrial designs demonstrate that, these two techniques help achieve superior DRC heatmap prediction performance with noisily labeled data.

- Routing-Free Crosstalk Prediction: Increasingly-smaller interconnect spacing in advanced technology nodes makes crosstalk a significant contributor to signal integrity and timing. We propose an ML-based routing-free crosstalk prediction framework. Given a placement, we identify routing-related features, along with electrical and logical features, which affect crosstalk-induced noise and delay. We then employ ML techniques to train the crosstalk prediction models, which can be used to identify crosstalk-critical nets in placement stages. Experimental validation on 12 benchmark circuits shows that the proposed method can clas-

sify more than 70% of crosstalk-critical nets after placement with a FPR of less than 2%. The computation speed is two orders of magnitude faster than a conventional method based on global routing.

- EDA Flow Parameter Tuning: A large spectrum of parameters available in EDA tools provides designers with a large freedom for achieving various design tradeoffs. The corresponding complexity, however, hinders designers from navigating towards optimal solutions. We propose a multi-stage automatic flow tuning tool, named *FlowTuner*, for efficient and effective parameter tuning of VLSI design flow. It utilizes both *exploitation* using transferred parameter knowledge from archived data from legacy designs and *exploration* via a multi-stage cooperative co-evolutionary framework. Furthermore, novel flow jump-start and early-stop techniques are developed to reduce the overall runtime for tuning. In addition, a concurrent multi-trade-off learning technique is proposed to enable truly multi-objective tuning by facilitating efficient Pareto front exploration. Experiments through a design flow using commercial tools have demonstrated that considerably better design outcomes are achieved in 50% shorter turnaround time compared against recent techniques.

Table 1.1: Example cases of ML for VLSI Design Automation.

Problem	Type	ML in./out.	Main customization
DRC hotspot	Pred.	Inside	Data augmentation, neural network architecture
DRC heatmap	Pred.	Inside	Integration of deep neural network with stochastic models
Crosstalk	Pred.	Inside	Feature extraction
Flow parm. tun.	Opt.	Outside	Integration of evolutionary algorithms,stochastic methods and other optimization techniques

The rest of this dissertation is organized as follows. Chapter 2 briefly introduces the background on ML algorithms to help understand our works. Details of our proposed solutions to DRC hotspot prediction, DRC heatmap prediction, routing-free crosstalk prediction and EDA flow

parameter tuning are presented in Chapter 3, Chapter 4, Chapter 5 and Chapter 6, respectively. Chapter 7 concludes this dissertation.

2. BACKGROUND IN MACHINE LEARNING

A popular definition of ML is that, ML is a class of artificial intelligence techniques that can discover knowledge from data and exploit the knowledge to improve the performance at tasks of interest [60]. As reviewed in [12], ML thrives in recent years and numerous ML techniques have been developed.

According to whether data is labeled or not, ML techniques are usually categorized into three classes, i.e., supervised learning, unsupervised learning and reinforcement learning. Supervised ML algorithms learn from correct input-output pairs, or called labeled data. Unsupervised ML algorithms learn from unlabelled data. In reinforcement learning, no labeled data is required. Instead, the machine is provided a way to quantify its performance in the form of a reward function to let it learn from the interaction with an environment. In this dissertation, supervised ML algorithms are utilized for prediction of DRC hotspot, DRC heatmap and crosstalk, while our EDA flow tuner learns from the interaction with EDA flows, i.e., in a reinforcement learning manner.

According to how knowledge is represented and discovered, Pedro Domingo [61], a professor of computer science and engineering at the University of Washington, categorizes main-stream ML algorithms into five tribes:

- **Symbolists**: This paradigm is based on high-level interpretation of knowledge and problems. It believes that "all intelligence can be reduced to manipulating symbols". Learning means to discovering rules to manipulate symbols. An advantage of this paradigm is that it can learn from data of different types and ensemble knowledge from different sources due to its high-level interpretation of knowledge and problems. Machine learning with decision trees is one representative algorithm in this paradigm. As shown in Figure 2.1, knowledge is represented by a set of binary decision making (symbols) in decision tree models. Strong learners can be obtained by ensembling simple decision trees. Random forest [62] and gradient boosted decision trees (GBDT) [63] (shown in Fogure 2.2) are two popular ensemble models. In

random forest models, decision trees are used as parallel learners and each tree is fit to a set of bootstrapping samples taken from the original dataset. Bootstrapping means randomly selecting samples from the original dataset with replacement. The final prediction result is obtained by averaging the results of decision trees. In GBDT models, each decision tree is fit to the residuals from previous ones and the final result is obtained by summing up results of all trees.

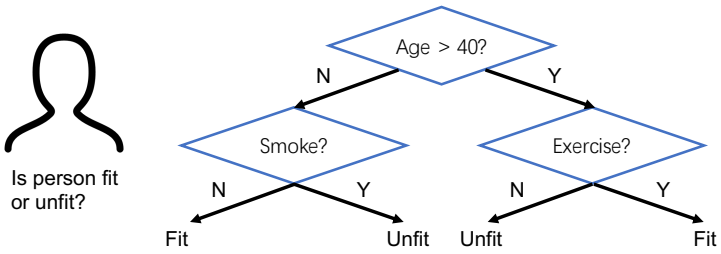


Figure 2.1: Illustration of decision tree models.

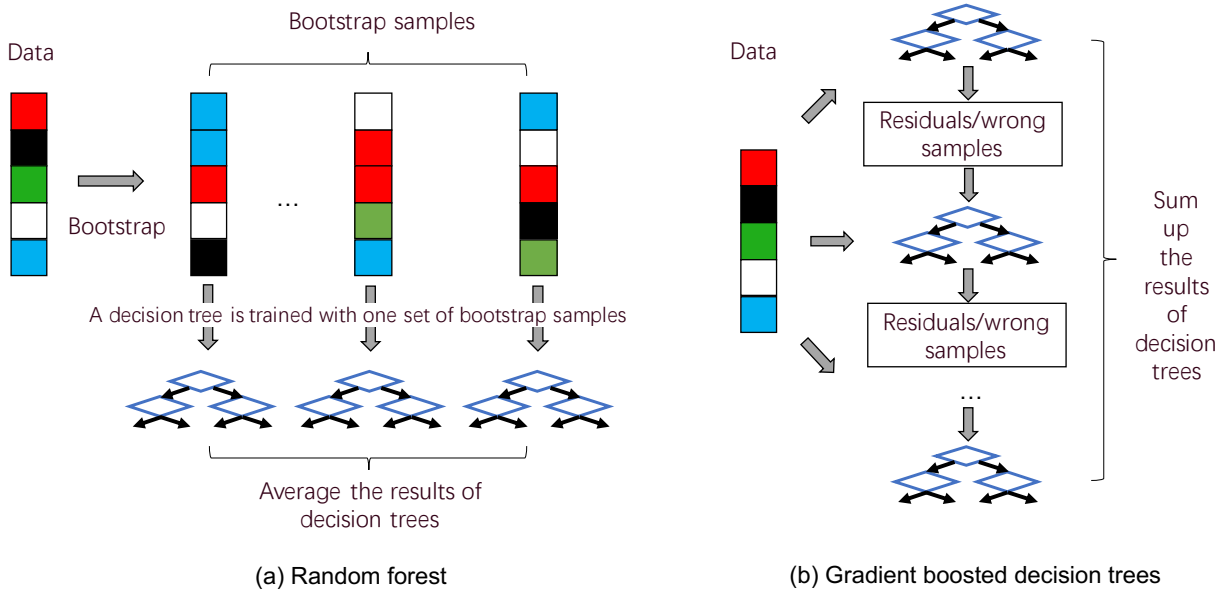


Figure 2.2: Illustration of the random forest model and the GBDT model.

- **Connectionists**: This paradigm emulates the brain and believes knowledge can be represented as connections between the neurons and the weights associated with connections. Figure 2.3 shows a neural network with fully-connected layers. Learning is viewed as updating the weights of connections and a representative algorithm is the backward propagation.

  In recent years, deep learning with neural network has achieved great success in processing of image, natural language, bioinformation, etc. Convolutional neural network (CNN) [64] is a popular framework for image classification, whose structure is shown in Figure 2.6. It is mainly composed by multiple layers of convolution, ReLU and pooling. A convolution layer consists of a set of trainable filters (kernels) sliding over the whole image, as shown in Figure 2.4. The step of sliding is called *stride*. A pooling layer, also referred to as down-sampling layer, applies a max/mean/average filter to the input, with a stride same as the filter size. The left side of Figure 2.5 shows an example of max-pooling, after which the resolution is reduced by a half. ReLU stands for the rectified linear activation function, which outputs the input directly if the input is positive; otherwise, it outputs zero. A popular framework for semantic segmentation is Fully Convolutional Network (FCN) [65], as shown in Figure 2.7. An FCN mainly consists of multiple layers of convolution, ReLU, pooling and transposed convolution operations, but without fully-connected layers. The right side of Figure 2.5 is an example of transposed convolution, which realizes up-sampling. Since all FCN's component layers do not require their inputs to be fixed sizes, FCN can take an image with variable size as input.

- **Evolutionaries**: This paradigm emulates the biological evolution on earth and believes learning can be conducted in a trial-and-error and following-the-winner manner. Knowledge is contained in the current best solution(s). Representative algorithms include genetic algorithm, simulated annealing, ant colony optimization (ACO) [66], etc. This paradigm is applicable to a wide range of optimization problems. It requires few information about the optimization objectives, for instance, no gradient information is needed.

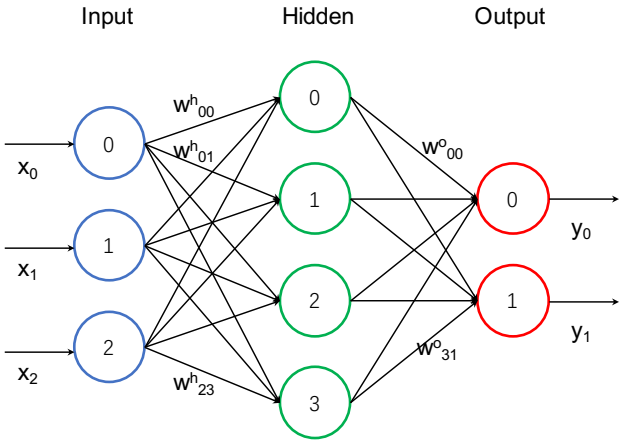


Figure 2.3: A neural network with fully connected layers.

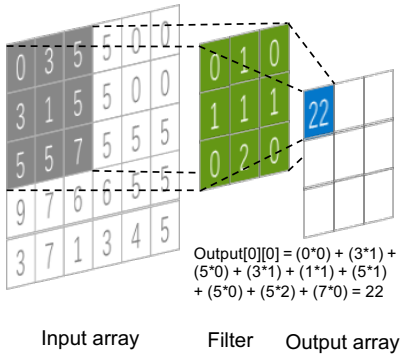


Figure 2.4: Examples of convolution operation.

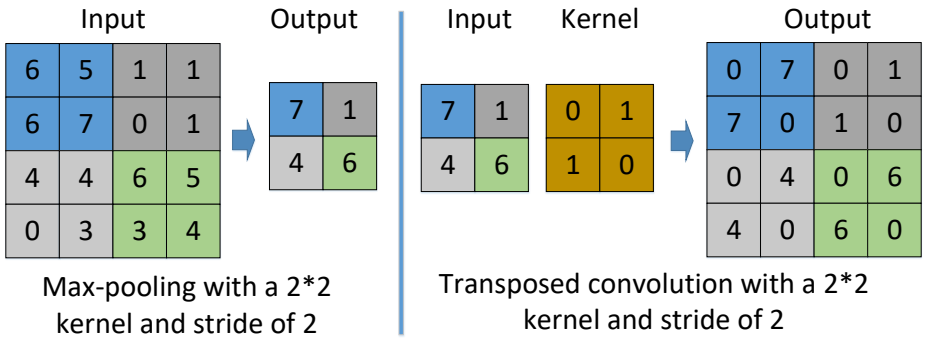


Figure 2.5: Examples of max-pooling and transposed-convolution operation.

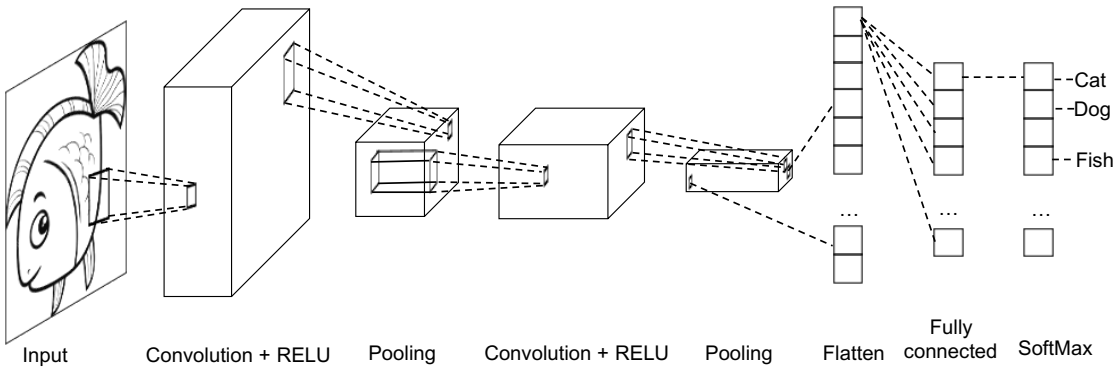


Figure 2.6: Example of a convolutional neural network.

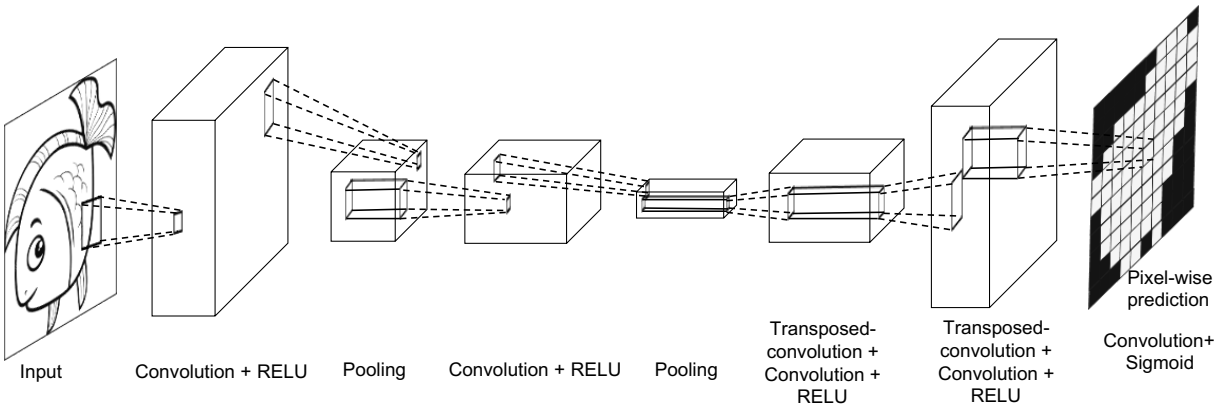


Figure 2.7: Example of a fully convolutional network.

Figure 2.8 depicts the concept of ACO. ACO is inspired by how ants find short paths from their nest to foods. Initially, ants randomly select paths to follow. And they lay down pheromone along the paths. For short paths, ant can go through them to the food and then go back to their nest quickly. Hence, the pheromone levels in shorter paths grow faster. In turn, the higher pheromone levels in shorter paths attract more ants, consequently leading to even higher pheromone levels. Eventually, almost all ants will follow the shortest paths. As we can see, ACO solves problems in a positive-feedback manner.

- **Bayesians**: This paradigm origins from theories of statistics. Knowledge is represented as statistical distribution of random variables and learning is regarded as determining the

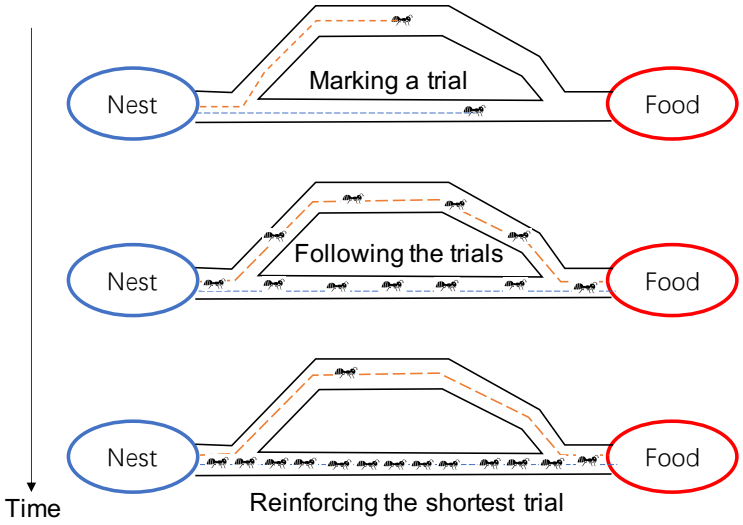


Figure 2.8: Illustration of ant colony optimization (adapted from [3]).

distributions that are most compatible with data. A representative learning algorithm in this paradigm is to use Bayes' theorem to determine the conditional probability of a hypotheses given some observations. This paradigm has solid theoretical foundation and provides a formalism to understand and quantify the uncertainty.

- **Analogizers**: This paradigm follows the "nearest neighbor" principal. The philosophy behind is that if two instances agree with one another in some respects, they will probably also agree in others. Learning is regarded as discovering proper metrics to measure the degree of agreement/similarity between instances. K-mean clustering (Figure 2.9), kernel-based support-vector-machine, matric-decomposition recommendation are examples of the analogizers' approach. The paradigm is applicable to problems ranging from unsupervised to supervised learning.

Table 2.1 summarizes the knowledge representation, advantages and representative algorithms of the five tribes of ML. It can be seen that each tribe has its strengths and is suitable for certain types of problems. Note that the tribes have no clear boundaries and there are efforts to integrate the advantages of multiple tribes of techniques [61]. To achieve best performance in EDA tasks, techniques from more than one tribe are deployed and integrated as needed in this dissertation.

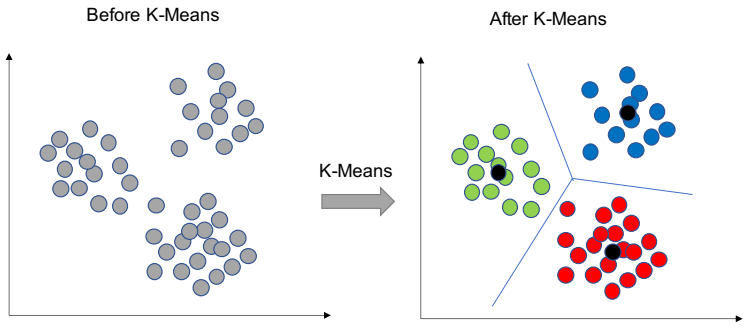


Figure 2.9: Illustration of the K-Means learning.

Table 2.1: Characteristic of five tribes of machine Learning.

	Knowledge repre.	Advantages	Repre. algo.
Symbolists	Rules to manipulate symbols	Cope with data of different types and ensemble knowledge	Decision tree
Connectionists	Connections between the neurons and the weights on them	Exploit properties of natural signals	Neural network
Evolutionaries	Current best solution(s)	Suitable for optimization problems	Genetic algorithm
Bayesians	Joint distribution of random variables	Cope with uncertainty	Bayes classifier
Analogizers	Metrics to measure the degree of similarity between instances	Cope with un-/semi-supervised learning	K-mean clustering

3. DRC HOTSPOT PREDICTION AT SUB-10NM PROCESS NODES USING CUSTOMIZED CONVOLUTIONAL NETWORK *

3.1 Introduction

Freedom from Design Rule Violation is a fundamental requirement for chip designs. Yet, it is challenging to achieve at sub-10nm process nodes due to a compounding effect arising from multiple factors. First, design rules at advanced technology nodes are immensely complex. Second, increasingly high chip complexity results in very dense devices on silicon die and therefore makes routing resource very scarce. Last but most importantly, cell pins are very difficult to access due to compact cell layout and cause the well-known pin accessibility problem [67]. In general, DRVs in chip layout are not known until detailed routing is completed. However, it is often hard to fix all DRVs by routing alone and cell placement needs to be modified as well. As such, it is important to predict DRC hotspots in early layout stages, such as cell placement, and take preemptive measure to avoid excessive iterations back and forth between placement and routing.

Closely related to DRVs, routability and routing congestion predictions have been extensively studied in the past [37, 68–70]. However, routability is not necessarily equivalent to freedom of DRVs as there can be DRVs after routing completion. Also, a congested layout does not necessarily mean it is not routable, although the two issues are correlated. In general, routing congestion can be seen during or after global routing, routability can be confirmed at detailed routing and DRVs are clearly known after design rule checking. Therefore, the predictions of congestion, routability and DRVs are progressively more difficult. There are a few works that predict routability in terms of DRVs [34, 71, 72]. However, these works did not explicitly emphasize pin accessibility, which is a main cause for many DRVs in nanometer designs. The pin accessibility issue has been investigated in [67, 73, 74], however, they are mostly focused on pin accessibility alone while ignoring other types of DRVs. The most recent work on pin accessibility prediction is restricted to only M2 short.

However, M2 short does not always dominate DRVs and there are many other types of DRVs that need to be considered as well. Our investigation on an industrial design shows that M2 short accounts for only about 20% of total DRVs. In the predictions, another important issue is whether or not global routing information is required as global routing is time consuming and makes the prediction difficult to be used frequently. Among existing works, only those for relatively easy congestion prediction [37] and specific kinds of DRV prediction [72, 74] can be performed without global routing while general DRC hotspot predictions [34, 71] still rely on global routing.

In this work, to the best of our knowledge, we introduce the first general DRC hotspot prediction technique that emphasizes pin accessibility and does not rely on global routing. This is a machine learning approach based on convolutional network. A key difference from other machine-learning-based predictions [34, 37, 71, 72, 74] is that a customized convolutional network architecture, J-Net, is developed as opposed to a plugin use of existing machine learning engines. This customization is used to address the mixed input resolution issue. The pin shape features for emphasizing pin accessibility has a resolution several orders of magnitude higher than other layout features such as pin density. Simply scaling low resolution features into high resolution would evidently cause inefficiency in computing. The proposed J-Net architecture can be automatically tuned for various feature resolutions from different designs. We demonstrate the effectiveness of proposed approach on industrial designs at 7nm process node. In a relatively relaxed yet realistic setting, the proposed J-Net achieves 93% true positive rate (TPR) while the TPR of the previous work [34] is only 56%, and the TPRs of the extensions of works in [37] and [74] are 52% and 79%, respectively, at about the same false positive rate. In a stricter setting, it can achieve 78.5% true positive rate (with false positive rate being 8.9%) when performing predictions on unseen designs. This true positive rate is 37% and 40% higher than those of [34] and [37] at about the same false positive rate, respectively.

The rest of this chapter is organized as follows. Section 3.2 briefly introduces the background on semantic segmentation to help understand our method. Relation between our work and previous ones is described in Section 3.3 and the details of proposed customized convolutional network for

DRC hotspot prediction is presented in Section 3.5. Section 3.6 shows our experimental setups and results. Section 3.7 concludes this chapter.

3.2 Background on Semantic Segmentation

Our prediction method is a customization of an existing semantic segmentation technique. The background on semantic segmentation is briefly introduced here in order to help understand our method. Different from image classification, whose output tells the class of a given input image, semantic segmentation outputs the class for every pixel. As such, the output of semantic segmentation can be treated as another image with the same size and resolution as the input image.

As introduced in Chapter 2, a popular framework for semantic segmentation is Fully Convolutional Network. FCN organizes the order of different layers according to the so-called encoder-decoder architecture. An encoder maps the raw input to compact feature representations, while a decoder processes feature representations to produce an output. Since all its component layers do not require their inputs to be fixed sizes, FCN can take an image with variable size as input.

U-Net [75] is a variant of FCN and designed for applications with relatively small amount of training data. It mainly consists of three parts, i.e. the encoding path, the decoding path and the bottleneck unit connecting two paths, as shown in Figure 3.1. The encoding path consists of repeated downing-sampling units, denoted by DOWN in Figure 3.1. The decoding path consists of repeated up-sampling units denoted by UP in Figure 3.1. At each up-sampling unit, the up-sampled map is concatenated with the feature map from the short-cut. The bottleneck unit is simply the stacking of a few convolution layers. Short-cuts exist at every resolution level and help U-Net capture both local and global information. The U-Net architecture resembles the multi-grid method for solving partial differential equations [76] and multilevel optimization in EDA [77], where a system is coarsened for several iterations, the main computation is performed at the coarsest level, and then the coarse solution is iteratively refined to the original granularity level.

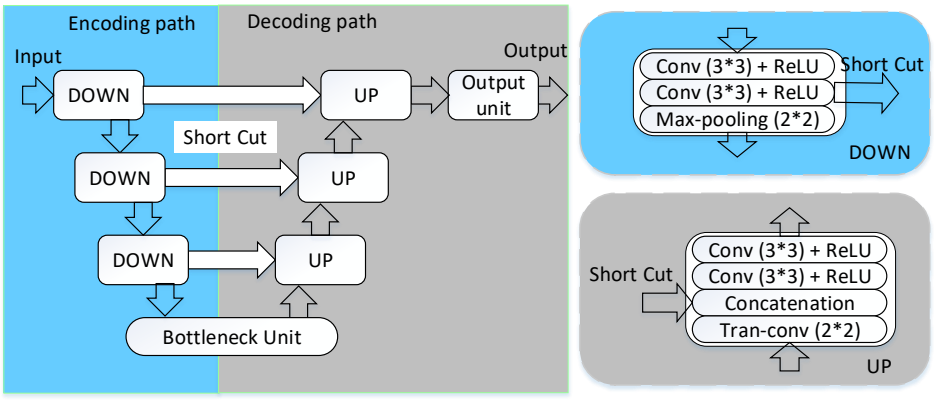


Figure 3.1: U-Net architecture (Reprinted with permission from [4]).

3.3 Relation with Previous Works

Early related works were mostly focused on routing congestion prediction. There are generally two types of approaches. One is analytical estimation based on net locations, such as [68, 69]. Such approach is fast to compute, but it can be very inaccurate and is hardly adopted in industry. The other is trial routing, which is much more accurate than analytical estimations but very time consuming. The pin accessibility problem was considered in several placement works [67, 70, 73], where an accessibility is evaluated by some simple score functions. This is very helpful for cell placement to address the pin accessibility problem but does not consider routability problems caused by congestion.

Recently, machine learning techniques were investigated to obtain accurate yet fast prediction on routability and/or DRVs [34,37,71,72,74,78,79]. The technique of [71] is based on support vector machine and requires global routing information as the machine learning model input. A neural network approach is proposed in [72] for predicting only short violations. In [34], a Convolutional Neural Network (CNN) model without using global routing is developed for classifying the overall number of DRVs and an FCN [65] model with global routing as input is designed for DRC hotspot prediction. A neural network ensemble approach for DRC hotspot prediction is described in [78]. However, it still requires global routing information as an input. Another neural network approach

that does not rely on global routing is reported in [79]. However, this method predicts only the total number of DRVs. In [37], conditional generative adversarial network is explored for routing congestion prediction without considering DRVs and this technique is validated on FPGA designs. A CNN-based pin accessibility prediction technique is introduced in [74]. However, it focuses only on M2 short violations and not for general DRVs. This very specific application allows it to use only two adjacent standard cells as the model input. Such small receptive field can easily fail for general DRVs, which may be caused by complex joint effects in a large region. The work in [74] can be extended to take information of a larger region (patch) into account to tell whether the tile is a DRC hotspot, as in [71, 72]. However, in the work [74], mixed resolution input channels have to be mapped and concatenated to a one-dimension feature vector, which then is processed by a sequence of fully connected layers. Such feature fusion scheme would become inefficient when a large patch size is used so that the concatenated feature vector would become very long and large fully connected layers have to be employed.

Machine learning-based DRC hotspot prediction can be realized with two different strategies. One is to perform semantic segmentation on entire layout like in [34] and our work. Alternatively, one can build a model for classifying a single tile (or global routing cell) using this tile and its neighboring tiles as input features [71, 72, 74]. This approach invokes a small model many times for all tiles and thus the feature data of one tile is fetched many times. By contrast, the feature data of one tile is fetched only once in semantic segmentation, and the computation among different patches can be highly amortized over the overlapping regions of those patches.

Overall, general DRC hotspot prediction without global routing information is still far from a solved problem. Our work is the first systematic study on this subject, to the best of our knowledge. Compared to most of previous machine learning-based works, which are plugin use of existing machine learning techniques with parameter tuning, our work is a customized machine learning approach to address mixed resolution issues caused by coarse-grained routing congestion and fine-grained pin accessibility.

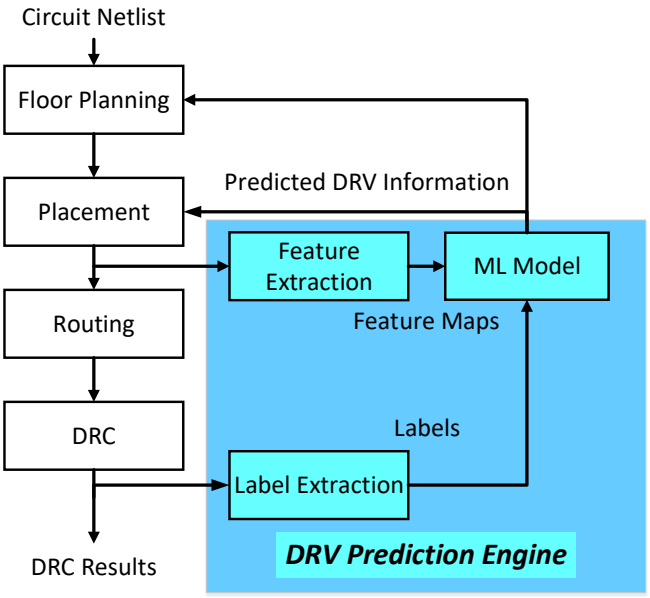


Figure 3.2: DRC hotspot prediction flow.

3.4 Problem Formulation

The DRC hotspot prediction flow is shown in Figure 3.2. The input features are extracted from the netlist of a given circuit and its placement solution. Labels are extracted from post-routing DRC results. To evaluate routability in terms of DRVs, a layout is tessellated into an array of uniform tiles, each of which is an $l \times l$ square[†]. The tile size is similar to that of a global routing cell (gcell). Then, a rectangular layout with size $W \times H$ is divided into $w \times h$ tiles, where $w = W/l$ and $h = H/l$. We target solving the following task at the placement stage:

- **DRC hotspot prediction.** Predict a binary DRC hotspot map $Y_{hotspot} \in \{0, 1\}^{w \times h}$ for each layout. Each entry in $Y_{hotspot}$ corresponds to one tile in the layout. "1" indicates there are DRVs in the tile and "0" indicates no DRV. In our experimental data, there are about 5% tiles belonging to DRC hotspots.

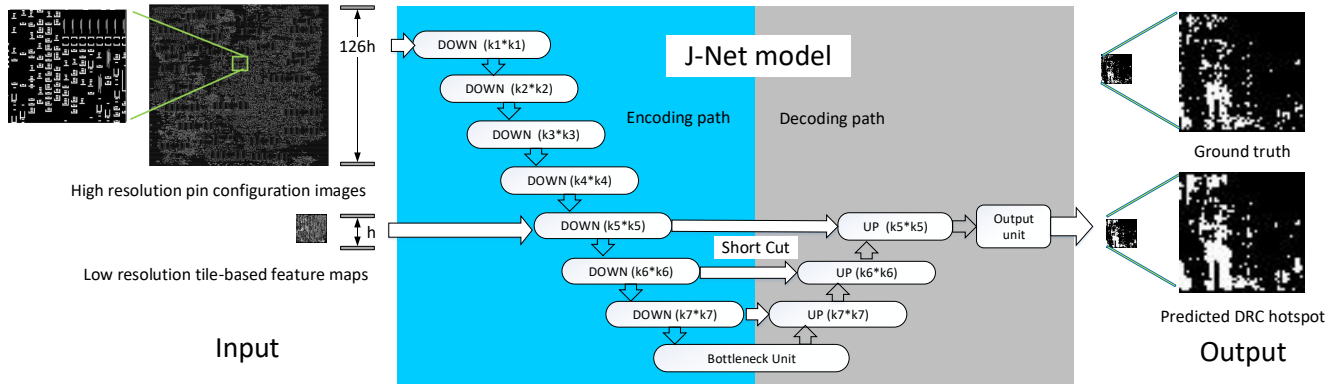


Figure 3.3: DRC hotspot prediction via J-Net (Reprinted with permission from [4]).

3.5 Customized Convolutional Network for DRC Hotspot Prediction

3.5.1 Overview and the Mixed Resolution Issue

We propose a customized convolutional network architecture, called J-Net, for DRC hotspot prediction with an explicit emphasis on pin accessibility. There is a key difference between our DRC hotspot prediction and conventional semantic segmentation. The output of semantic segmentation is an image of the same or a bit lower resolution as the input image. In our case, the image resolution for pin accessibility is about two orders of magnitude finer than the hotspot map. The resolution of pin image features must be a half wire pitch or finer to ensure the recognition of a pin shape. The hotspot map is based on routing tiles and a tile typically covers 20 routing tracks or more. Therefore, a tile easily contains thousands of pin image pixels. Directly plugin use of U-Net requires us to adopt very high resolution hotspot maps, which cost a large and unnecessary computing overhead. Moreover, we need to consider tile-based features and hence features with huge resolution difference need to be simultaneously input to the model. The mixed resolution issue motivates us for a customized approach.

[†]In our experiments, l is set to be $1.26\mu m$.

3.5.2 J-Net Convolutional Network Architecture

The proposed J-Net is an extension of the U-Net architecture [75] and can handle highly different resolutions between different input channels, and between input and output. When all input channels and the output have the same resolution, J-Net degenerates to U-Net.

J-Net follows the typical encoder-decoder architecture, as illustrated in Figure 3.3. The encoding path consists of repeated down-sampling units, which are indicated by *DOWN* in Figure 3.3. Similarly, the decoding path is formed by a sequence of up-sampling units. For simplicity, we refer to the kernel size of max-pooling layer in a down-sampling unit as the kernel size of the down-sampling unit, and to the kernel size of transposed convolution layer in an up-sampling unit as the kernel size of the up-sampling unit. The $k * k$ in parenthesis in Figure 3.3 represent the kernel sizes of the down-sampling and up-sampling units. Compared to U-Net, four main modifications are made in J-Net.

1. Input channels of different resolutions are fed into different levels at the encoding path. If an input channel is fed to a middle level of the encoding path, it is concatenated with the feature representations of the same resolution produced from the previous upper level. By doing so, J-Net can easily accommodate input channels with highly different resolutions.

2. The number of decoder levels is less than that of encoder. Therefore, the output resolution can be significantly lower than the top input channel at the encoder path. This configuration makes the encoder-decoder architecture look a flipped letter "J" and this is why this architecture is called J-Net.

3. The number of convolution operations in each down-sampling/up-sampling unit is reduced from 2 to 1. This change greatly decreases the number of trainable parameters as well as the need for large amount of training data. This is particularly appealing in EDA applications, where training data is difficult to obtain.

4. The kernel size of down-sampling/up-sampling units in U-Net is usually fixed at 2×2. By contrast, J-Net treats the kernel sizes as variable parameters and can automatically tune their

values for various input channel resolutions caused by different process nodes and feature selections.

3.5.3 Automatic Kernel Size Tuning and Branch Path Addition

In constructing a J-Net, the number of encoding path levels and the kernel sizes of its down-sampling units are automatically tuned. The tuning is to ensure that the feature representations from different input channels have the same resolution when they are concatenated at certain level. Sometimes, such tuning is insufficient and a new branch needs to be added to the encoding path. We illustrate the tuning and branch addition of two cases through the examples as follows.

- Case 1: The resolution of one input channel is the divisor of the other channel. For example, channel 1 has resolution 12600×12600 and channel 2 has resolution 100×100. The relative resolution between them is $12600/100 = 126$. Prime factorization is performed on the relative resolution to obtain $126 = 7 \times 3 \times 3 \times 2$. Then, four down-sampling units are generated between channel 1 and channel 2 as shown in Figure 3.3. And the kernel sizes are arranged in descending order, i.e., $k_1 = 7, k_2 = 3, k_3 = 3$ and $k_4 = 2$. With this tuning technique, the resolution of high-resolution features is gradually reduced to the same as that of low-resolution features such that they can be concatenated at certain level. The other down-sampling units have their kernel sizes as the default value “2" of U-Net or other empirical values.

  Generating as many down-sampling layers with small kernel sizes as possible between mixed-resolution input channels, in contrast to inserting one layer with a large kernel size, enables our J-Net model to learn hierarchical features easily, leading to high prediction accuracy. Arranging kernel sizes of down-sampling layers in the descending order helps reduce our model's memory usage in training and inference. In the above example, the resolutions of intermediate feature representations after each of four down-sampling layers are $\frac{12600\times12600}{k_1^2}$, $\frac{12600\times12600}{k_1^2k_2^2}$, $\frac{12600\times12600}{k_1^2k_2^2k_3^2}$, and $\frac{12600\times12600}{k_1^2k_2^2k_3^2k_4^2}$, respectively. Arranging the kernel sizes (i.e., k_1 to k_4), in descending order helps reduce the

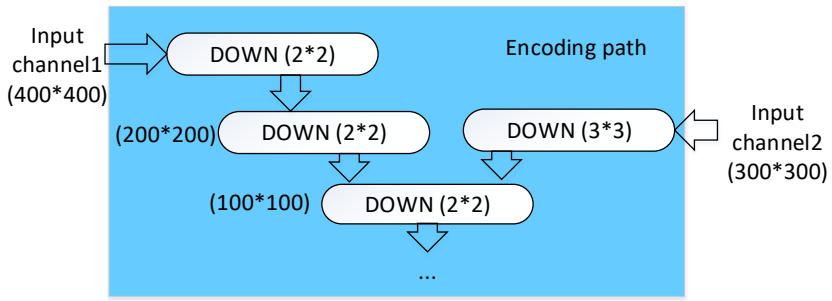


Figure 3.4: An example of encoding-path for two input channels of case 2 (Reprinted with permission from [4]).

size of intermediate feature representations, leading to less memory usage.

- Case 2: No input channel has resolution as divisor of the other channel. For example, channel 1 has resolution 400×400 and channel 2 has resolution 300×300. In this case, we first find the greatest common factor of the resolutions of the two channels, which is 100 in this example. Next, we compute the relative resolutions of the channels to the greatest common factor, which are 4 and 3 here. Then, factorization is performed to obtain $4 = 2 \times 2$ and $3 = 3$. As a result, there are two branches forming an encoding tree as shown in Figure 3.4, as opposed to the single encoding path in Figure 3.3.

Mathematically, there could be other cases. For example, two input channels with resolutions that prime to each other, such as 11×11 and 97×97. However, it is extremely rare that such strange cases can happen in practice.

3.5.4 Feature Selection

3.5.4.1 Pin Configuration Images

Complicated patterns formed by nearby pins and routing blockages have great impact on local routability involving standard cells at advanced technology nodes. In order to capture the effects of such patterns, pin configuration images are generated for each layout, which have sufficiently high resolution to accurately capture the location and shapes of pins as well as routing blockages

of the entire layout. An example of pin configuration image is shown in Figure 3.3. With proper alignment, each pixel in the image represents either an empty region or a region covered by pins or blockages. Empty regions are associated with “0" grey level, and regions covered by pins are associated with “100" while those covered by blockages are associated with “200". Pins and blockages are assigned with different grey scales as they play different roles in pin accessibility. A relatively large pin makes it more accessible while a large blockage degrades pin accessibility. To allow machine learning models to identify complex patterns formed by pins located at different metal layers, one pin configuration image is generated for each layer where pins reside. Four images are generated for each design in total for our benchmark designs.

3.5.4.2 Tile-based Features Maps

Besides the pin configuration images, some netlist and layout features are also extracted for each tile (or global routing cell). Then, the feature values of all the tiles are combined into feature maps according to their locations in the layout, and employed as inputs to J-Net. Extracted features include:

- Routing resource describing the percentage of a tile area that is occupied by IPs, which affects the number of available routing tracks in the tile. For example, a pixel of value “25" in the routing resource feature map represents a tile whose 25% area is covered by IPs.
- Connection features. In total, we deploy three tile-based connection features, i.e., number of local nets, number of global nets and RUDY [68]. Local nets are nets whose pins are completely inside a tile, while global nets are those which connect to pins outside the tile. Each pixel value in the local net feature map represents the number of local nets in the corresponding tile. Similarly, each pixel value in the global net feature map represents the number of global nets in the tile. RUDY is a score function at each location obtained by overlapping net bounding boxes covering this location. The RUDY feature map is generated in the same way as that introduced in [34].

For the technology node and test cases in our setup, each tile is $1.26\mu m \times 1.26\mu m$ large,

and corresponds to 126×126 pixels in a pin configuration image. Hence, the resolution of pin configuration images are 126 times of tile-based feature maps. Both pin configuration images and tile-based feature maps are generated based on the information collected at the placement stage, via our self-developed scripts.

Note that the relative resolutions between different input channels are fixed after training the J-Net model. However, thanks to the properties inherited from the FCN architecture, the trained J-Net model can be applied to placement instances of various sizes, as long as the resolutions (not necessarily the size) of input channels are the same as those used in the training process.

3.5.5 Data Augmentation

Due to the difficulty for obtaining real-world 7nm designs, the amount of available training data is quite limited. We employ the following data augmentation techniques in the training process to address this problem.

- Cropping layouts into small ones. For large layouts, we crop their tile-based feature maps and pin configuration images into smaller ones with a sliding window. The size of the window is 50-tile-by-50-tile. On one hand, the window size is sufficiently large such that this cropping does not significantly affect the model training. On the other hand, the amount of training data is effectively increased. The stride of sliding window is $40\times$ single-tile-length, slightly less than the length of the sliding window. As such, the cropped feature maps (so as the pin configuration images) have small overlap with each other. By allowing the overlap, more training data can be obtained. This augmentation technique can increase the number of training samples by about 10 times, since many designs in our benchmarks contain more than 150-by-150 tiles. After cropping, the size of tile-based feature maps becomes 50-by-50, while the size of pin configuration image becomes 6300-by-6300.

- Random flipping: In the training process, every time a data sample is fetched, random flipping is performed on it. The layouts after flipping should have the same routability as that before the flipping. However, the inputs to the model become different and can still enhance

the training.

3.6 Experiment

3.6.1 Testcases and Data Collection

Several industrial designs at 7nm process node were used as testcases. We generated multiple placement instances for each design by performing a state-of-the-art industrial physical synthesis flow with various (but still realistic) parameter settings. After placement, pin configuration images and tile-based feature maps were generated. Then, all placement solutions were routed and design rule checking was performed to obtain the ground truth DRC hotspot maps, which were used as model training labels.

Table 3.1: Testcase characteristics (Reprinted with permission from [4]).

Design	Chip Size (μm^2)	#IPs	#Gates	#Nets
D1	59.52×62.21	0	13569	15552
D2	59.52x×129.60	0	10796	17843
D3	211.20×391.39	0	328611	338846
D4	79.68×84.24	0	18283	25068
D5	122.88×233.28	0	68393	86640
D6	122.88×95.90	0	46001	49337
D7	138.24×80.35	0	35627	38456
D8	284.16×95.90	0	100566	108187
D9	76.80×401.76	1	52659	70338
D10	249.60×316.22	3	149456	191513
D11	280.32×489.89	16	81384	105678
D12	253.44×671.33	28	200454	247271

In total, there are 12 designs in the testcase suite, four of which have IPs (or macros). Table 3.1 summarizes the characteristics of the testcases. By varying optimization settings for each design, we obtained 166 placement instances in total (before applying the data augmentation technique introduced in Section 3.5.5). It is noteworthy that placement instances with a wide range of the number of DRVs can be obtained by varying optimization settings. Taking design D12 as an

example, among 13 generated placement instances, the number of DRV hotspot tiles varies from 2614 to 11506. The average and the standard deviation of the number of DRV hotspot tiles were 7361 and 3258, respectively.

3.6.2 Comparison Setup

3.6.2.1 Impact of features

To analyze the impact of extracted features on prediction performance, we tested several different combinations of the following features in evaluating J-Net and U-Net.

- **H**: high resolution pin configuration images.
- **R**: routing resource features as described in Section 3.5.4.
- **Cn**: connection features as described in Section 3.5.4.
- **Cg**: congestion map based on the global routing results.
- **D**: density features such as logic gate pin density, clock pin density, logic cell density, filler cell density, etc.

3.6.2.2 Effectiveness of Proposed DRC Hotspot Prediction Solution

The effectiveness of the proposed techniques was evaluated through comparisons with recent previous works [34, 37, 74], various machine learning models and combinations of different features.

In the experiment, we compared five different machine learning models, three of which are based on recent previous works [34, 37, 74].

- **J-Net.** This is our proposed model.
- **U-Net.** Direct plugin use of the U-Net architecture [75]. As U-Net restricts that all input channels and the output have the same resolution, only tile-based features are accommodated here.

- **FCN.** The Fully Convolutional Network approach in [34]. The features used are the same as in [34].

- **cGAN.** The conditional Generative Adversarial Net approach is an extension of the work in [37]. The work was for routing congestion prediction for FPGA designs, which is different from our hotspot map prediction. We did our best to implement as close to [37] as possible. GAN is notoriously difficult to train and we adopted the techniques described in [80] to improve the training.

- **CNN.** The Convolutional Neural Network approach is an extension of the work in [74]. In contrast to considering only the information in a small region covering two nearby cells as in [74], the implemented CNN model takes the pin configuration images covering one single tile and the corresponding tile features as input to classify one tile each time.

As in [34], the FCN features include R, Cg, D and RUDY [69]. Similar to [37], the features of cGAN contain R, D and connectivity images, which is a set of straight lines connecting pins of the same net. The input of CNN includes H, R and Cn.

3.6.2.3 Performance of Variants of J-Net Architecture

In our benchmarks, the resolution of the pin configuration images is 126 times of that of tile-based features maps. There are many options in determining how many down-sampling units are generated between the high-resolution- and low-resolution- inputs as well as their kernel sizes. We compared the performance of our proposed J-Net architecture with a few variants.

- **J0.** This is our proposed version. Four down-sampling units are inserted between the pin configuration images and the tile-based feature maps. And their kernel sizes are 7, 3, 3 and 2 respectively.

- **J1.** Four down-sampling units are inserted. But the kernel sizes are in ascending order, i.e., the kernel size for the first unit is 2 while the size for the fourth unit is 7.

- **J2.** Only one down-sampling unit, with kernel size 126, is generated.

To show the memory efficiency of J-Net brought by flexibly handling highly different resolutions, we scaled low resolution tile-based feature maps into high resolution and implemented the following methods:

- **J3.** J-Net with high resolution inputs. Inputs include pin configuration images as well as upsampled tile-based feature maps. Proposed kernel size tuning technique is deployed. Note that the output is low resolution tile-based DRC hotspot maps, which makes this J-Net model still has higher memory efficiency than standard U-Net models whose input and output have the same resolution.

3.6.3 Training and Testing Schemes

For the labelled data that we can obtain, two different partition schemes between training and testing sets were applied for the model evaluation.

- **Scheme 1**: This partition scheme was applied on the 166 placement instances. We randomly chose 80% of the placement instances as training data, which were further increased by the data augmentation techniques described in Section 3.5.5. The other 20% of the placement instances were used for testing. Please note testing such placement instances may involve some same designs as those in the training set, although their placements are different. This scenario may practically exist in reality, where a few routing and DRC results of a design have already been obtained before applying machine learning on new placement solutions of the same design. Similar schemes have also been employed in previous works [37, 71].

- **Scheme 2**: Twelve rounds of experiments were performed and their results were averaged. In each round, all placement instances from 11 designs were taken as training data and applied with data augmentation. Placement instances of the only remaining design were used as testing data. Each round had a distinct testing design. This is a stricter testing scheme than scheme 1 as not only the testing placement instances but also the testing design are completely unseen in the training.

In our data set, only a small portion (about 5%) of the tiles belong to DRC hotspots. In other words, it is a highly imbalanced data set. Weighted loss function, i.e. giving a heavy weight to DRC hotspots during the computation of training loss, as the one used in [71], was employed to address this problem.

3.6.4 Performance Metrics

Due to the large imbalance between the number of positive samples (hotspots) and negative samples (non-hotspots), simple tile-wise accuracy is no longer an effective measure of prediction performance. For example, consider a case where 90% samples are positive and a model simply predicts all the samples are positive. In this case, accuracy of 90% is obtained although the model almost does nothing useful. In our experiment, the following metrics were employed to evaluate the effectiveness of different approaches.

1) True Positive Rate (TPR), False Positive Rate (FPR), Precision and F1-score: Table 3.2 summarizes the calculation of TP, TN, FP and PN, based on which TPR (also known as recall), FPR, Precision and F1-score can be estimated.

- False positive rate: $\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP+TN}}$.
- True positive rate: $\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP+FN}}$.
- Precision $= \frac{\mathrm{TP}}{\mathrm{FP+TP}}$.
- F1 score: $\mathrm{F1} = 2 \times \frac{\mathrm{TPR} \times \mathrm{Precision}}{\mathrm{TPR+Precision}}$.

Table 3.2: Calculation of prediction accuracy (Reprinted with permission from [4]).

	Prediction Result	
Ground Truth	Positive	Negative
Positive	True Positive(TP)	False Negative(FN)
Negative	False Positive(FP)	True Negative(TN)

2) Area Under Curve (AUC): Receiver Operating Characteristic (ROC) curve is the TPR vs. FPR curve, which indicates the trade-off between TPR and FPR by varying the classification thresholds. This is similar to power-delay trade-off curves in chip designs. A large AUC of ROC-curve [81] implies better prediction performance. An AUC of ROC-curve of 1 means perfect prediction, and random guesses result in 0.5 AUC. AUC of Precision-Recall (PR) curve [82] is also a commonly used metric for evaluating classification performance. A larger AUC of PR-curve implies better prediction performance too.

3.6.5 Experiment Results

3.6.5.1 Results from Training & Testing Scheme 1

Effects of Different Features on J-Net and U-Net

The ROC and PR curves are shown in Figure 3.5 while the results on different metrics are listed in Table 3.3. One can see that pin configuration images have a great impact on prediction performance. Even when it is used as the only feature (J-Net(H)), the prediction performance is much better than employing numerous tile-based feature maps (U-Net(R+Cn+D)). Although the role of pin configuration maps is critical, the effect of tile-based features is also significant. For FPR around 10%, tile-based features can improve TPR from 83% to 93%. Overall, J-Net can achieve around 93% TPR when FPR is less than 10%. Another observation is that there is a large gap between global routing congestion map (Cg) and DRC hotspot in our testcases. When congestion map is used as the only feature (U-Net(Cg)), the TPR is as less than 30% when the FPR is 9.21%.

Note that using more features as input (compare J-Net (H+R+Cn) with J-Net(H+R+Cn+D+Cg)) does not necessarily enhance the prediction performance, since it might induce the over-fitting problem, given the limited amount of training data.

Comparison Among Different Models and with Prior Arts

The ROC and PR curves of different models are displayed in Figure 3.6 and the numerical results are listed in Table 3.4. Among the models, FCN is based on the previous work [34], and

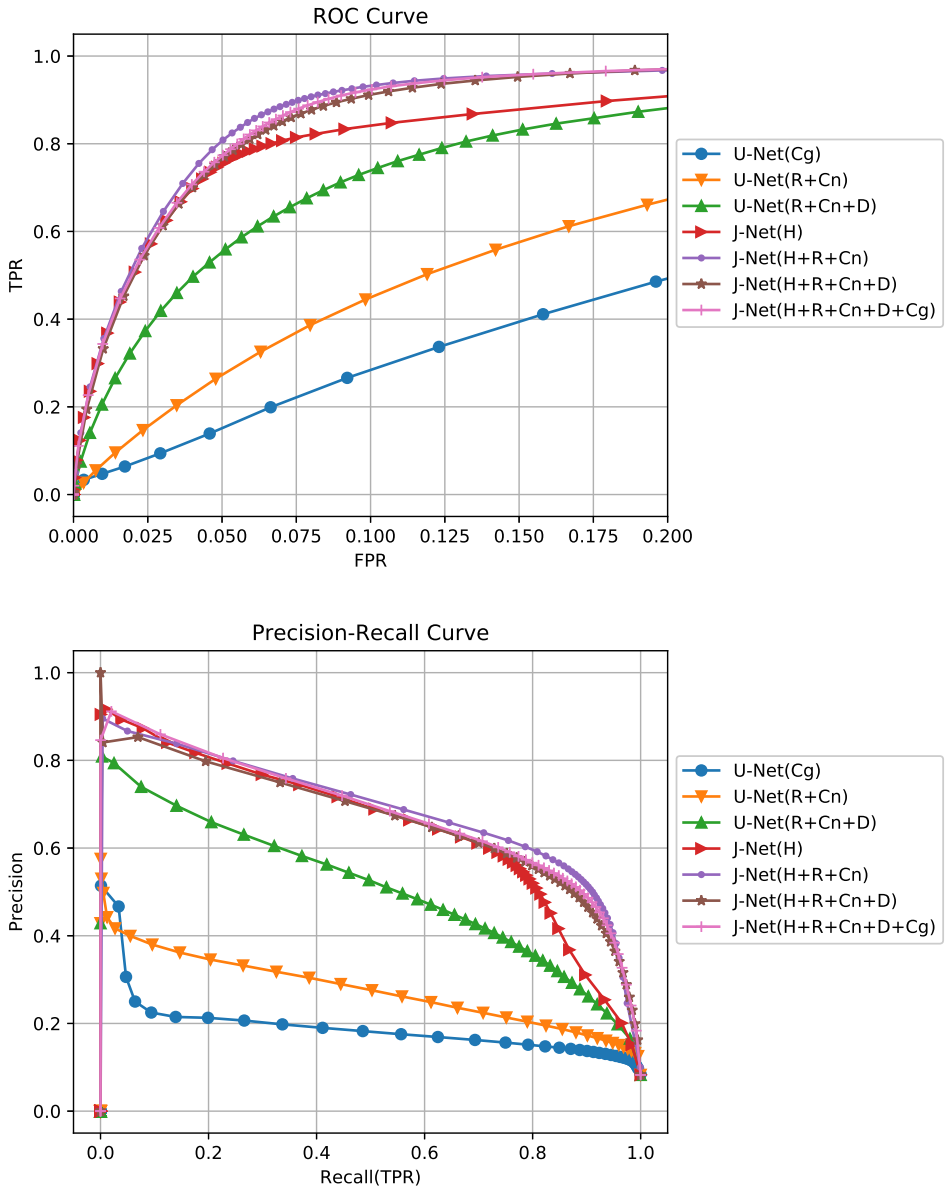


Figure 3.5: ROC and PR curves for different features (scheme 1) (Reprinted with permission from [4]).

cGAN and CNN are extensions of the works in [37] and [74], respectively. The prediction performance of J-Net is evidently superior to other models. At FPR of around 10%, J-Net achieves TPR of 93%, while the TPRs of FCN, cGAN and CNN are only 56%, 52% and 79%, respectively. One important reason for the inferior performance of the FCN [34] and the cGAN [37] is that they do not consider the pin-accessibility issue, which is a significant cause of DRVs at 7nm process

Table 3.3: Results from different features for U-Net and J-Net (scheme1) (Reprinted with permission from [4]).

			FPR$\leq 5\%$				FPR$\leq 10\%$			
	ROC	PR	FPR	TPR	precision	F1	FPR	TPR	precision	F1
U-Net(Cg)	0.738	0.192	4.58%	13.91%	21.48%	16.89%	9.21%	26.61%	20.66%	23.26%
U-Net(R+Cn)	0.835	0.277	4.79%	26.40%	33.21%	29.42%	9.83%	44.48%	28.98%	35.09%
U-Net(R+Cn+D)	0.913	0.509	4.58%	52.94%	51.04%	51.97%	9.60%	72.89%	40.63%	52.18%
J-Net(H)	0.937	0.647	4.84%	74.67%	58.19%	65.41%	9.12%	83.34%	45.16%	58.58%
J-Net(H+R+Cn)	0.958	0.686	4.67%	78.64%	60.30%	68.26%	9.75%	92.98%	46.24%	61.77%
J-Net(H+R+Cn+D)	0.958	0.666	4.76%	75.19%	58.77%	65.97%	9.93%	91.12%	45.26%	60.48%
J-Net(H+R+Cn+D+Cg)	0.959	0.680	4.74%	75.77%	59.03%	66.36%	9.49%	91.72%	46.56%	61.77%

nodes. In contrast to considering only the information in two nearby cells as in [74], we extended the CNN model to take into account the information in one tile each time. But the prediction accuracy is still not as high as that reported in [74], for which one important reason might be that general DRVs rather than only M2 short are considered in our experiment. Using a larger patch size might help improve the prediction accuracy of the CNN approach, but significant overhead in run-time and memory-usage would be induced, as elaborated in Section 3.3.

Figure 3.7 shows the AUC of ROC curve and memory usage of four J-Net versions. We can find that the default J-Net architecture J0, version J1 and version J3 achieve the same (highest) AUC, while the performance of version J2 is obviously inferior to other architectures. Unlike J2 which down-samples the pin configuration images via one down-sampling unit with large kernel size, J0 and J1 down-samples the high-resolution images in four steps, which enables the models to learn hierarchical features, consequently helps improve prediction accuracy. Also, arranging the kernel sizes of down-sampling units in descending order reduces memory usage of J0 by 35.8% compared to J1. J3 scales low resolution tile-based feature maps into high resolution, resulting in 53.6% higher memory usage than J0.

3.6.5.2 Results from Training & Testing Scheme 2

For scheme 2, the comparisons for different features and different models are summarized in Table 3.5 and Table 3.6, respectively. Each data entry in the tables is the average result of 12 rounds. As scheme 2 is a stricter scenario than scheme1, all results degrade compared those of

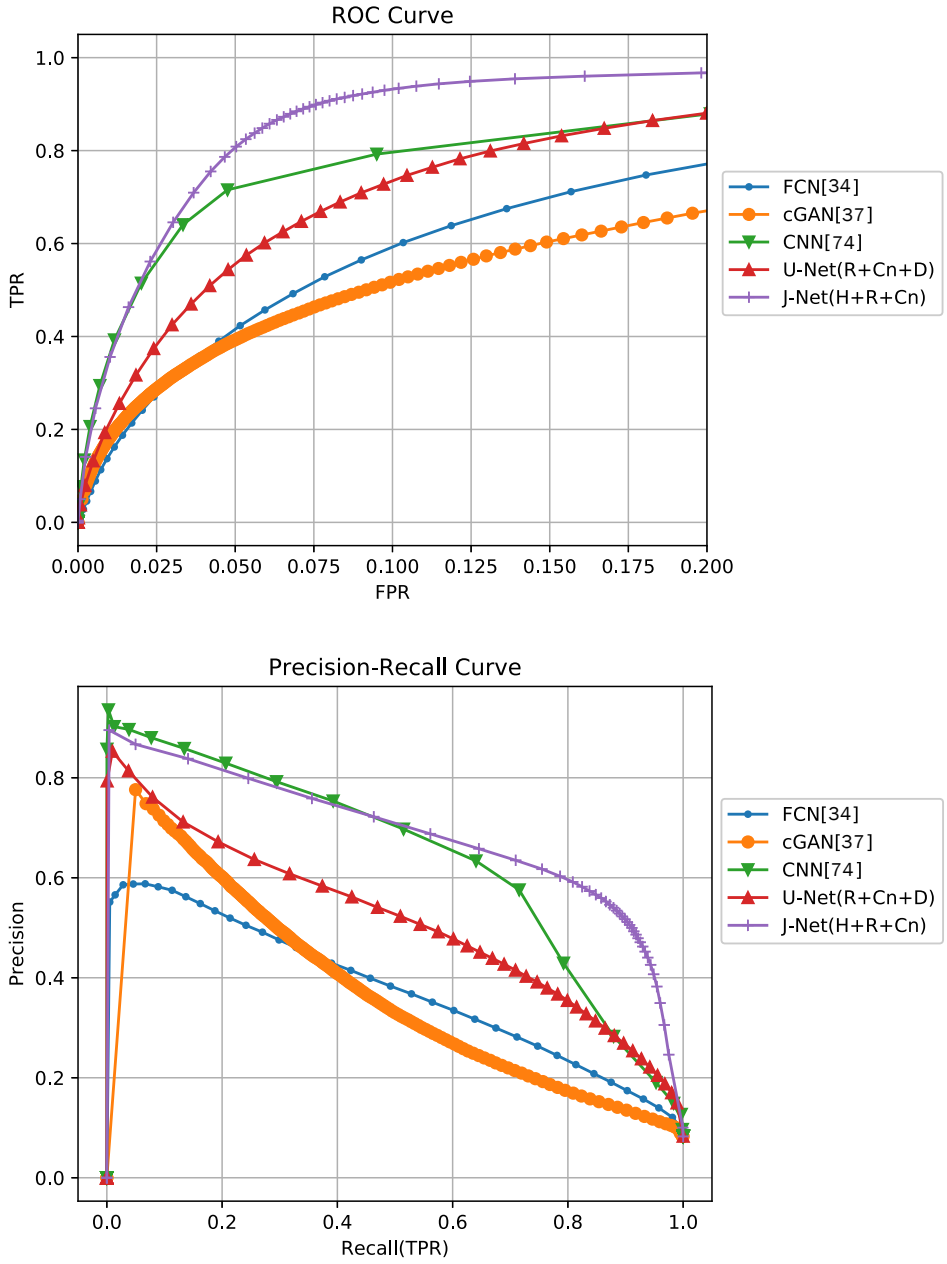


Figure 3.6: ROC and PR curves of different models and previous works (scheme 1) (Reprinted with permission from [4]).

scheme1. However, key observations are not changed here. First, pin configuration map is perhaps the most important feature. Second, J-Net significantly outperforms the other models including the previous works [34, 37]. Compared with FCN [34], cGAN [37] and U-Net(R+Cn+D), our J-Net(H+R+Cn) achieves 37%, 40% and 22% higher TPR, respectively, at similar FPRs.

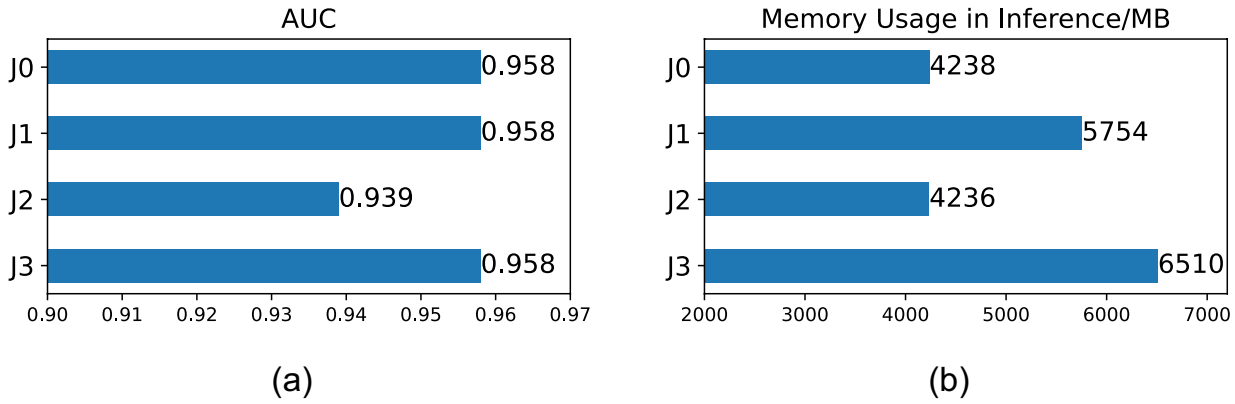


Figure 3.7: Performance and Memory Usage of variants of J-Net architecture (scheme1): (a) AUC of ROC curves, (b) memory usage in reference for a placement instance with 50×50 tiles (Reprinted with permission from [4]).

Table 3.4: Comparison among different models and with previous works (scheme 1) (Reprinted with permission from [4]).

Metric	FCN [34]	cGAN [37]	CNN [74]	U-Net	J-Net
AUC (ROC)	0.867	0.818	0.927	0.913	0.958
AUC (PR)	0.375	0.360	0.639	0.509	0.686
FPR	9.01%	9.93%	9.50%	9.60%	9.75%
TPR	56.48%	51.67%	79.2%	72.89%	92.98%
Precision	35.12%	31.94%	42.9%	40.63%	46.24%
F1-score	43.31%	39.48%	55.7%	52.18%	61.77%
Need GR?	Y	N	N	N	N

Table 3.5: Results from different features(scheme 2) (Reprinted with permission from [4]).

	AUC(ROC)	AUC(PR)
U-Net(R+Cn)	0.731	0.250
U-Net(R+Cn+D)	0.854	0.415
J-Net(H)	0.883	0.582
J-Net(H+R+Cn)	0.913	0.604
J-Net(H+R+Cn+D)	0.900	0.586
J-Net(H+R+Cn+D+Cg)	0.911	0.599

Table 3.6: Comparison among different models and with prior arts (scheme2) (Reprinted with permission from [4]).

Metric	FCN [34]	cGAN [37]	U-Net	J-Net
AUC (ROC)	0.788	0.714	0.854	0.913
AUC (PR)	0.279	0.287	0.415	0.604
FPR	9.10%	9.69%	9.47%	8.90%
TPR	41.04%	38.1%	56.08%	78.47%
Precision	31.28%	29.9%	35.80%	46.16%
F1-score	32.25%	29.9%	39.32%	53.95%

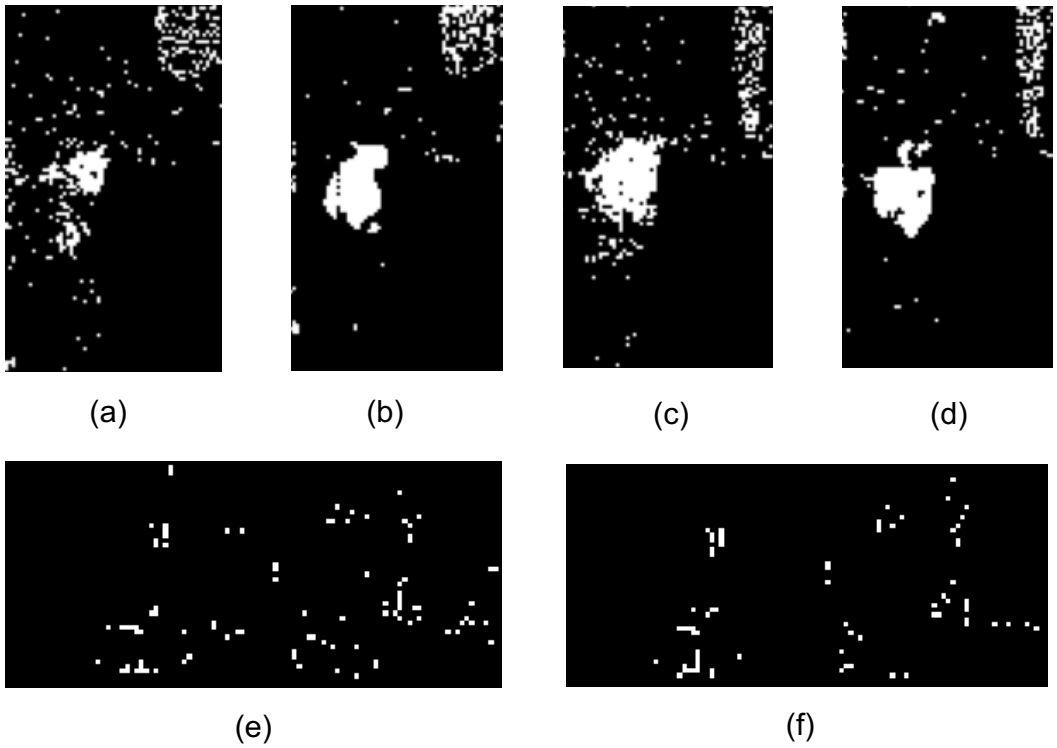


Figure 3.8: Snapshots of DRC hotspot prediction results (scheme 2). Regions in white represent DRC hotspots. (a), (c) and (e) are DRC hotspot ground truth; while (b), (d) and (f) are corresponding prediction results.

3.6.5.3 Training, Feature Extraction and Inference Time

For each round of training with 20 epochs, the training time of FCN, U-Net or cGAN on a GeForce RTX 2080-Ti GPU is less than 1 hours while training time of J-Net or CNN is around

30 hours. Since the model can be reused repeatedly for different and unseen designs of the same technology node, the amortized training cost is limited. For feature extraction and inference time, the dominating factor is whether or not global routing information is required. Consider circuit D8 as an example, which has about 101K gates and 108K nets. Without global routing, the total run-time on feature extraction, data transformation and inference by each of cGAN, CNN, U-Net and our J-Net is less than 1 minute for one layout design. However, the global routing of D8 takes several hours and it is required by the FCN method [34].

3.7 Conclusion

A machine learning approach is proposed for predicting DRC hotspot at cell placement stage for sub-10nm designs. It is a customized convolutional network architecture that can simultaneously take high resolution pin images and low resolution placement information as input features. This approach is evaluated on 12 industrial designs at 7nm process node and compared with three recent works. The results show that its prediction performance considerably outperforms the previous works and has fast feature extraction and inference time as no global routing information is required.

4. INTEGRATING DEEP NEURAL NETWORK WITH STOCHASTIC MODELS FOR DRV HEATMAP PREDICTION

4.1 Introduction

Routability prediction in terms of DRVs has attracted a lot of attention recently [34, 36, 71, 72, 74, 83, 84]. However, these works formulate DRV prediction as a binary classification problem, i.e., only predicting whether DRVs will occur in each region, instead of regression that tells DRV density. Evidently, DRV density provides more detailed information than binary classification in guiding DRV mitigation techniques. For example, appropriate amounts of white space [85] can be allocated into different regions of the layout according to the predicted DRV density to improve routability of a placement solution. Multi-class classification is a middle ground between binary classification and regression. However, a multi-class classifier with a small number of class categories does not provide detailed DRV density information; while a large number of categories will make the model very difficult to train, especially with highly imbalanced DRV data, given that regions with DRVs accounts for only about 5% of entire layout area in several typical industrial designs and regions with high DRV density are even more scarce. DRV density regression has been studied in [86]. It utilizes a simple multivariate adaptive regression model and does not leverage the strong learning capability of modern ML techniques.

A challenging issue in DRV prediction is the randomness in the distribution of DRVs, which in essence is brought by the non-determinism in parallel detailed routing. Figure 4.1 shows the DRV maps of two routing solutions on the same placement generated by the same router and with the same settings. It can be seen that these two DRV maps are far from identical, though similar to each other. From the view of training machine-learning models, the DRV labels collected after routing are very “noisy". Even more complicated, the noise depends on the DRV density. As shown in Figure 4.1, the difference between these two maps is larger in regions with higher DRV density. Such noise will make the training process of DRV estimators unstable or converge to a

solution that overfits to the noisy data. Essentially, the issue of noisy label data caused by non-deterministic parallel computing is a general problem in applying ML techniques to EDA tasks, given that numerous EDA algorithms are accelerated by parallel computing. However, to our best knowledge, this problem has hardly ever been studied in the EDA field.

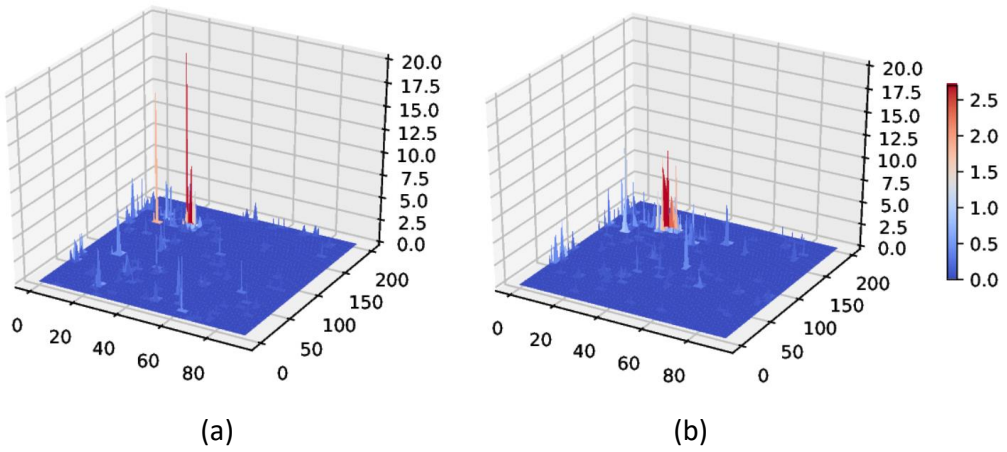


Figure 4.1: DRV maps from two routing rounds for the same placement solution and with the same routing tool.

To address this issue, we combine the strong modeling capability of deep neural network and the strength of stochastic models in coping with uncertainty. We regard that DRV maps from different routing rounds for one placement solution are governed by the same stochastic distribution, which accounts for the their similarity. And a deep neural network is deployed to model the mapping from a placement solution to the corresponding stochastic distribution of DRVs. Each DRV map from one routing round is viewed as one sample from the stochastic distribution. And the uncertainty in sampling causes noise in DRV maps. We propose to train the deep neural network model in a Maximum Likelihood Estimation manner: to find a set of network parameters that makes the network model compatible with observations, i.e., (placement, DRV map) pairs in a probabilistic sense.

In this work, we target at DRV density prediction at the placement stage, considering the non-determinism in routing. Our main contributions are:

1. The spatial distribution of DRVs in the layout is modelled by a Poisson point process as well as a Gaussian random field, which lays a theoretical foundation to address the issue of noisy DRV labels caused by non-deterministic parallel routing.

2. A novel focal likelihood loss function and the Gaussian random field layer technique are proposed based on the stochastic modelling of DRVs to emphasize the stochastic properties of DRV distribution.

3. We adapt the J-Net architecture proposed in Section 3.5 to model the mapping from a placement solution to the corresponding stochastic distribution of DRVs. And we further show that training the neural network with our proposed focal likelihood loss function essentially is a Maximum Likelihood Estimation approach.

4. Experiment results on industrial designs at a 7nm process node demonstrate that, the proposed method helps achieve superior DRV density prediction performance with noisy label data. Using the regression results for multi-class classification, the performance is evidently superior to directly training a multi-class classifier as in [84]. Our regression results can also be used for binary classification and achieve comparable accuracy to J-Net-based binary classifiers. It is noteworthy that the regression-based binary classification can train once and provide prediction results for various label thresholds; while binary classifiers have to retrain for different thresholds.

The remainder of the chapter is organized as follows. Problem formulation is presented in Section 4.2. Section 4.3 briefly introduces the backgrounds that are helpful for understanding our method. Details of the proposed DRV density prediction solution is presented in Section 4.4. Experiment setup and results are shown in Section 4.5 and Section 4.6, respectively. Section 4.7 gives some concluding remarks.

4.2 Problem Formulation

A layout is tessellated into an array of uniform rectangular tiles as described in Section 3.4. We target at solving the following DRV density prediction task at the placement stage.

- **DRC heatmap prediction**: Predict a real number map $Y_{heatmap} \in [0, K]^{w \times h}$ for each layout. Each entry in $Y_{heatmap}$ represents the the number of DRVs in the corresponding tile. K is the maximal number of DRVs in a tile. Figure 4.2 shows the frequency of DRV densities in tiles. We can see that the DRV densities in over 99.6% of tiles are in the range of $[0, 10]$. Hence, we set K to be 10, in our experiment. For tiles with #DRVs larger than 10, we cut-off their values to be 10.

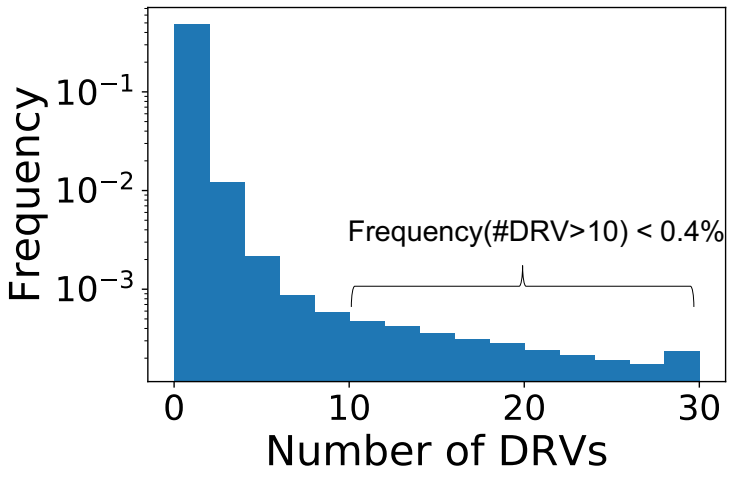


Figure 4.2: Frequency of DRV densities in tiles.

4.3 Preliminaries

4.3.1 Background on Poisson Point Process and Gaussian Cox Process

A point process [87] is a probabilistic model for random scatterings of points in some space S. A point process in space S is a Poisson point process with intensity function μ if for any

$B \subseteq S$ such that $\mu(B) = \int_B \mu(x)\, dx < \infty$, the number of points occurring in B obeys the Poisson distribution with mean $\mu(B)$, i.e., $Prob(N(B) = k) = e^{-\mu(B)} \frac{\mu(B)^k}{k!}, k = 0, 1, 2, \cdots$, where $N(B)$ is the number of points occurring in B. A Poisson point process is fully characterized by the intensity function μ. For any $B \subseteq S$ such that $\mu(B) < \infty$, $E(N(B)) = Var(N(B)) = \mu(B)$, where $E(*)$ and $Var(*)$ represent mean and variance, respectively.

The Poisson point process is a basic point process model, in the sense that it possesses the property of "no interaction" between points or "complete spatial randomness", while real world point patterns usually exhibit some degree of clustering or repelling. The Gaussian Cox process [88] is an extension of the Poisson point process. It consists of two components, i.e., an intensity function μ modeled by a Gaussian random field (GRF) [89] over some space S and a Poisson point process parameterized by the intensity function μ. The GRF enforces that the intensity function has similar values in nearby regions and it is a popular technique to capture the clustering behaviour of points.

4.3.2 Focal Loss

The *focal loss* (FL) [5] is designed to address the extreme class-imbalance during the training of classification models. For a sample with label $l \in \{0, 1\}$, the raw classification output $y \in [0, 1]$ can be regarded as the predicted probability of this sample with label "1". The focal loss is defined as:

$$FL(y, l) = -(1 - p_t)^\gamma log(p_t) \ , \tag{4.1}$$

$$p_t = \begin{cases} y, & if \quad l = 1 \\ 1 - y, & otherwise. \end{cases} , \tag{4.2}$$

where γ is a hyper-parameter.

The focal loss utilizes a modulating factor $(1 - p_t)^\gamma$ to down-weight well-classified samples. According the definition of p_t in Equation (4.2), we can find that p_t near 1 indicates samples that the model can correctly classify with high probabilities, i.e., easy-to-classify samples, whereas p_t near 0 means difficult to classify. When a sample has small p_t, the modulating factor is near 1 and the loss is unaffected. As $p_t \to 1$, the factor goes to 0 and the loss for well-classified samples

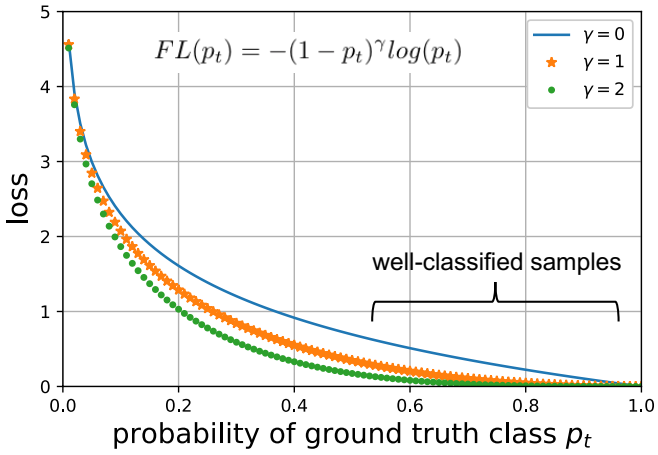


Figure 4.3: Illustration of focal loss (adapted from [5]). By setting γ to 0, focal loss degrades to standard cross entropy loss. Compared with cross entropy loss, setting $\gamma > 0$ down-weights well-classified samples, focusing more on difficult-to-classified samples.

is down-weighted. The parameter γ smoothly adjusts the rate at which easy samples are down-weighted. Figure 4.3 depicts how γ affects the behaviours of the FL. In our experiment, we set γ to be 1. Note that the FL can not be applied to regression problems directly.

4.4 Methodology

4.4.1 Adapting J-Net for DRC Heatmap Prediction

To adapt J-Net to DRC heatmap prediction, we employ a one-channel Sigmoid layer scaled by the factor K for the output unit in Figure 3.3, where K is set to be the maximal #DRVs in a tile. In other words, the output is a real number map $\in [0, K]^{w \times h}$.

4.4.2 Addressing the Non-determinism in Routing via Stochastic Modellings of DRVs

4.4.2.1 Stochastic Modelling of DRVs

We model the spatial distribution of DRVs in the layout as a Gaussian Cox process in a discrete two dimensional space. First, it has a heterogeneous Poisson point process characterized by a intensity function $\mu(i, j)$ over $1 \leq i \leq w$ and $1 \leq j \leq h$, where $\mu(i, j)$ represents the DRV intensity in the tile (i, j). *Heterogeneous* here means the $\mu(i, j)$ varies at different (i, j). According to the property of Poisson point process, the number of DRVs occurring in the (i, j)th tile obeys a

Poisson distribution with mean $\mu(i, j)$. Based on the property of Poisson distribution, $\mu(i, j)$ also represents the variance of #DRVs in the tile. Second, the intensity function $\mu(i, j)$ is governed by an Gaussian random field, which enforces that nearby tiles have similar DRV density.

To demonstrate the rationality of modelling the spatial distribution of DRVs as a Gaussian Cox process, we ran 32 rounds of routing for five different placement instances with the same router and the same settings, and analyzed the resulted DRV maps. When the number of DRVs in a tile is treated as a random variable, we obtained 32 samples for each of such random variables, and estimated their mean and variance based on the results of 32 samples. According to the theory of Poisson point processes, the mean of #DRVs in a tile is supposed to be equal to the variance. Figure 4.4 shows the relationship between the mean and the variance of #DRVs in a tile in our date set. We can find that the variance seems to be linear to the mean, which conforms to the characteristic of the Poisson point process. But the ratio of variance against mean is about 3, rather than 1. However, a relatively-small ratio of variance to mean shows that the Poisson point process is still a reasonable model for the distribution of DRVs. In addition, we can easily see clustering behaviours of DRVs from Figure 4.1. It motivates us to further capture the clustering behaviour of DRVs by a Gaussian random field.

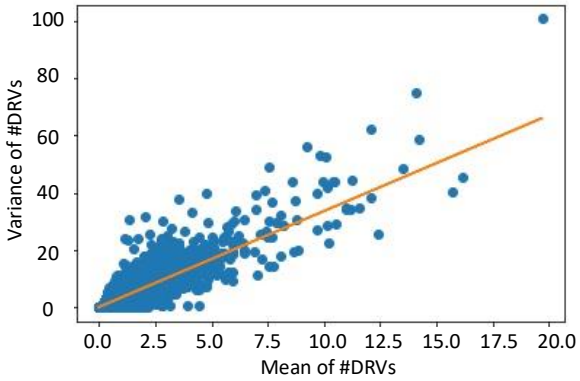


Figure 4.4: Mean VS Variance of #DRVs in a tile.

4.4.2.2 Focal Likelihood Loss (FLL)

Based on the aforementioned stochastic model of DRVs, we propose the FLL for the training of DRC heatmap predictor with noisy label data. The output of our DRC heatmap predictor $Y \in \mathbb{R}^{w \times h}$ is viewed as the intensity function that governs the Poisson point process for the distribution of DRVs in the layout. The noisy DRV density map $L \in \mathbb{N}^{w \times h}$ ($\mathbb{N}$ is the set of non-negative integers) collected from one round of routing is regarded as one sample generated from the Poisson point process. The training of the predictor is conduced in a maximum likelihood estimation manner: finding a set of network parameters that maximize the likelihood that the DRV density map L collected after routing are sampled from a Poisson point process with intensity function Y.

The FLL is defined as follows:

$$FLL(Y, L) = -\frac{1}{wh} \sum_{i \in [1,w], j \in [1,h]} (1 - P[L(i,j)|Y(i,j)]) log(P[(L(i,j)|Y(i,j)] + \varepsilon)\,, \tag{4.3}$$

$$P[L(i,j)|Y(i,j)] = \frac{Y(i,j)^{L(i,j)} e^{-Y(i,j)}}{L(i,j)!}\,, \tag{4.4}$$

where ε is a small positive number*. Essentially, Equation (4.3) is obtained via replacing the p_t in Equation (4.1) by $P[L(i,j) \mid Y(i,j)]$, which is the probability that the $L(i,j)$ is a sample generated from a Poisson distribution with mean $Y(i,j)$. The FLL guides the training process to converge to a solution that maximizes the likelihood $log(P[(L(i,j)|Y(i,j)]$. The term $1 - P[L(i,j) \mid Y(i,j)]$ is the focal loss term, which down-weights easy samples adaptively during training, as introduced in Section 4.3. The introduction of the focal loss term helps the FLL handle the issue of imbalanced data set.

An outstanding characteristic of our proposed FLL is that it penalizes prediction results according to the likelihood, rather than the absolute errors. Consider two cases. In the first case, the predicted intensity $Y(i,j)$ is 0, while the #DRV label in this tile $L(i,j)$ is 1. In another case, the predicted intensity $Y(i,j)$ is 9, while the #DRV label is 10. Since the absolute prediction errors in

*ε is set to be $10e^{-6}$ in our experiments.

both cases are 1, mean-square-error-based (MSE-based) loss functions will produce the same loss for both cases. But our FLL will penalize the first case much more heavily, since $L(i,j) = 1$ can never happen if the Poisson distribution has mean 0. As shown in Figure 4.1, the absolute difference between two different routing solutions for the same design is larger in regions with higher DRV density. As such, we prefer to tolerate larger absolute prediction errors for tiles with higher DRVs density. Our proposed FLL can handle such requirement nicely.

To reduce runtime overhead during training, we pre-calculate $log(k!)$ for every integer $k \in [0, K]$ before training and store them in a table. Then the log-likelihood can be easily calculated as:

$$log\, P[L(i,j)|Y(i,j)] = L(i,j)log(Y(i,j)) - Y(i,j) - log(L(i,j)!)\,, \tag{4.5}$$

where $log(L(i,j)!)$ is obtained via the look-up table. Next, the probability $P[L(i,j)\,|\,Y(i,j)]$ can be obtained by performing an exponentiation operation.

4.4.2.3 Gaussian Random Field Layer

The GRF can be implemented as a network layer that processes the unstructured prediction output $Y \in \mathbb{R}^{w \times h}$ of the J-Net model and outputs a structured output $Z \in \mathbb{R}^{w \times h}$, as shown in Figure 4.5. Here, unstructured output means the prediction of intensity function without considering the dependency among tiles, while the structured output means the prediction taking into account the dependency relationship.

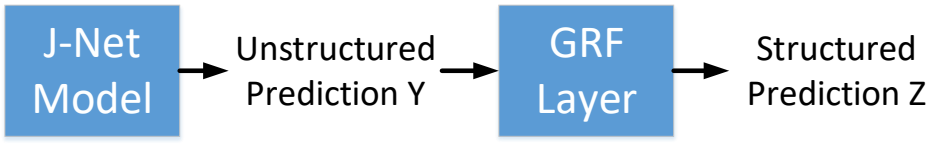


Figure 4.5: Role of the Gaussian random field Layer.

Essentially, the GRF layer is composed of a couple of matrix operations. Denote the flattened

vector of Y as y, which is a vector of length $w \times h$, and the flattened vector of Z as z. The GRF layer outputs a z that maximizes the probability

$$Prob(z \mid y) == \frac{1}{\alpha}\text{exp}\{-E(z,y)\}, \tag{4.6}$$

where α is a normalization term ensuring that the integral of the probability over z is equal to 1, and $E(z,y)$ is the energy at (z,y), which is defined as:

$$E(z,y) = \sum_i (y_i - z_i)^2 + \beta \sum_{i<j} S_{ij}(z_i - z_j))^2\,, \tag{4.7}$$

where β is a weighting factor and $S_{ij} \geq 0$ is the similarity measure between the i-th and the j-th entry. The first term on the right hand side of Figure 4.7 enforces the structured output Z to be close to the unstructured output Y, while the second term enforces similar entries to have similar output. The similarity among entries is described by the similarity measure S_{ij}.

The similarity measure S_{ij} is defined based on the proximity of two tiles as follows.

$$S_{ij} = \begin{cases} \text{exp}\{-\text{dist}(i,j)/\sigma\}, & i \neq j \\ 0, & i = j \end{cases}\,, \tag{4.8}$$

where dist(i,j) denotes the Manhattan Distance between two tiles, and σ is a learnable parameter.

$E(z,y)$ can be rewritten in a vector form as follows:

$$E(z,y) = z^T Q z - 2z^T y + y^T y \propto z^T Q z - 2z^T y\,, \tag{4.9}$$

$$Q(i,j) = \begin{cases} 1 + \beta \sum_{h \neq i} S_{ih}, & i = j \\ -S_{ij}, & i \neq j \end{cases}\,. \tag{4.10}$$

We let the weighting factor β as a learnable parameter.

Once the values of β and S_{ij} are obtained, Q can be computed. For positive-definite matrix Q,

the optimal structured output z^* that maximizes $Prob(z \mid y)$ could be obtained by

$$z^* = Q^{-1}y\,, \tag{4.11}$$

where $(*)^{-1}$ represents matrix inversion.

With the GRF layer, the training procedure becomes a two-stage process. At the first stage, the GRF layer does not come into play and the J-Net part is trained solely to output unstructured prediction. At the second stage the GRF layer is added to the model as the last layer and the entire model is trained end-to-end to deliver structured output.

4.4.3 Discussions

4.4.3.1 Why to use a Gaussian Cox process?

There are mainly two reasons for modelling the spatial distribution of DRVs in the layout space as a Gaussian Cox process:

1. The Gaussian cox process is a good match to the distribution of DRVs. To be specific, the domain of a Gaussian Cox process is the set of non-negative integers, which matches the range of #DRVs in the layout naturally. In addition, as discussed in Section 4.4.2.1, we observe a linear correlation between the variance of #DRVs in a tile with its mean, and the clustering behaviours of DRVs in the layout, which match the properties of the Gaussian Cox process.

2. Our focal likelihood loss function and the Gaussian random field layer technique based on the Gaussian Cox model can be implemented efficiently. To be more specific, as introduced in Section 4.4.2.2, the likelihood based on a Poisson point process can be calculated via the look-up table, multiplication and accumulation operations. Also, the Gaussian random field layer can be implemented via matrix operations as introduced in Section 4.4.2.3.

4.4.3.2 *How to integrate machine learning techniques with stochastic models to handle the non-determinism in other applications?*

- Step 1: Build a stochastic model for the target of interest, maybe guided by the domain knowledge and statistical analysis results.
- Step 2: Develop ML techniques for the estimation of the parameters that govern the stochastic model of the target of interest. Utilize our proposed likelihood-based loss function during training.

4.5 Experiment Setups

4.5.1 Testcases

Experiments were conducted on 12 industrial designs, same as those used for DRC hotspot prediction experiments (Section 3.6). Placement instances from randomly-chosen 8 designs were taken as training data while placement instances of remaining 4 designs were utilized as test data. Please note that, testing designs were totally unseen during training.

4.5.2 Comparison Setup and Performance Metrics

4.5.2.1 *Experiment 1: DRC Heatmap Regression*

We compared our proposed method with Zhou et al. [86] and a naive extension of the J-Net method, i.e., the MSEL method.

1. Zhou [86]. This is a multivariate adaptive regression-based method using as input the pin density, the ratio of blocked areas, the HPWL density and global routing results. Since Zhou [86] did not introduce the loss function used, we deployed the naive mean-square-error loss function.
2. **MSEL.** This is a J-Net-based regression method with the mean-square-error loss function.
3. **J-Net Plus.** Regression model with our proposed FLL and the GRF layer techniques.

The following performance metrics were utilized:

1. The mean-square-error.

2. The Pearson correlation coefficient between the DRC heatmap label data and the prediction result.

3. The mean of normalized errors for tiles with label $\#DRV > 0$. For a tile with label $l \in \mathbb{R}$ and prediction output $y \in \mathbb{R}$, the normalized error is defined as

$$\frac{|l - y|}{\sqrt{l}} , \tag{4.12}$$

   where $|x|$ represents the absolute value of x. According to the properties of Poisson processes, the standard deviation of the DRV density is equal to the square root of the mean. Hence, $\sqrt{l}$ can be viewed as an approximate of the standard deviation of the DRV density. The normalized error measures how far from the predicted DRV density to the label normalized against the standard deviation of the DRV density.

4. The mean of the predicted DRV density for tiles with label $\#DRV = 0$. For perfect prediction, the mean is 0. The smaller the mean, the better the prediction performance.

5. To evaluate the benefit of DRV density prediction in guiding DRV mitigation techniques, we make the following assumption: allocating k units of white space to a tile with l DRVs can eliminate $min(k, l)$ DRVs. Here we adopt a simple white space allocation strategy according to the DRV density prediction results: letting $k = round(y)$, where $round(y)$ represents the rounded value of the predicted DRV density y. The total units of allocated white space and the number of eliminated DRVs were calculated and compared.

4.5.2.2 Experiment 2: Using Regression Results for Multi-Class Classification

Multi-class classification result can be obtained via setting thresholds for the regression result. In our experiment, tiles are divided into three classes according to their #DRVs, using 2 and 6 as the label thresholds. Tiles with less-than-2 DRVs are easy to route while those with greater-than-6

DRVs are hard to route. The regression-based classification results were compared with the results of a J-Net-based multi-class classifier (denoted as **MC**), during the training of which a weighted MSE loss was adopted to address the imbalanced data issue. As for performance evaluation, we calculated the confusion matrix and compared the accuracy for each class.

4.5.2.3 Experiment 3: Using Regression Results for Binary Classification

Regression results can also be transformed into binary classification results. For label thresholds 2, 4 and 6, we compared the regression-based classification results with results of J-Net-based binary classifiers. To visualize the binary classification performance, we plotted the ROC curves of different methods, which describe the trade-off between TPR and FPR (introduced in Section 3.6).

4.6 Experiment Results

4.6.1 DRC Heatmap Regression

Table 4.1: Regression results.

			Ours
Metric	Zhou [86]	MSEL	J-Net Plus
Mean-square-error	1.118	0.874	0.542
Pearson correlation coefficient r	0.296	0.366	0.666
Mean of normalized error for tiles with #DRV>0	1.080	0.865	0.716
Mean of predicted DRV density for tiles with #DRV=0	0.306	0.254	0.179
Units of allocated white space	431825	253852	265258
Number of eliminated DRVs	112649	72866	131562

Regression results of different methods are shown in Table 4.1. We can find that, compared with Zhou [86]'s method and a naive extension of the J-Net method, our proposed J-Net Plus method has much lower mean-square-errors, higher correlation coefficient, lower normalized errors for tiles with #DRV>0 and lower predicted DRV densities for tiles with #DRV=0 as well. It means that, with a highly-imbalanced and "noisy" data set, our proposed techniques guide the training process of regression models to find better solutions.

Table 4.2: Multi-class classification results.

		Prediction Result			
Method	Label	0	1	2	Acc.
MC	0 (#DRV<2)	**1102471**	71549	3032	94%
	1 (2≤#DRV<6)	12921	**15708**	1229	53%
	2 (6≤#DRV)	5437	858	**4295**	41%
				Ave	**63%**
J-Net Plus	0	**1102472**	71831	2749	94%
	1	7571	**20762**	1525	70%
	2	169	4063	**6358**	60%
				Ave	**75%**

Figure 4.6 shows the snapshot of DRC heatmap prediction results of proposed J-Net Plus method for three testing samples. In general, the proposed method predicts the locations as well as #DRV fairly well. The prediction results look like a somewhat smoothed version of the labels.

In terms of the benefits of DRV density prediction results in guiding the DRV mitigation techniques, based on the assumption made in Section 4.5, prediction results of our method can help eliminate 131562 DRVs out of 234758 ones by allocating 265258 units of white space. In contrast, the MSEL method can only help eliminate 72866 DRVs with a similar amount of white space; while Zhou [86]'s method method can help eliminate 112649 DRVs with 63% larger amount of white space compared with our method.

4.6.2 Using Regression Results for Multi-Class Classification

Table 4.2 shows that confusion matrices for multi-class classification and the prediction accuracy for each class. It can be seen that proposed J-Net Plus method has obviously higher accuracy than the MC method. Besides, for those misclassified tile samples by the J-Net Plus method, most of them are classified to be categories similar to their label categories. In contrast, many misclassified samples by the MC method are classified to be categories far away from their labels.

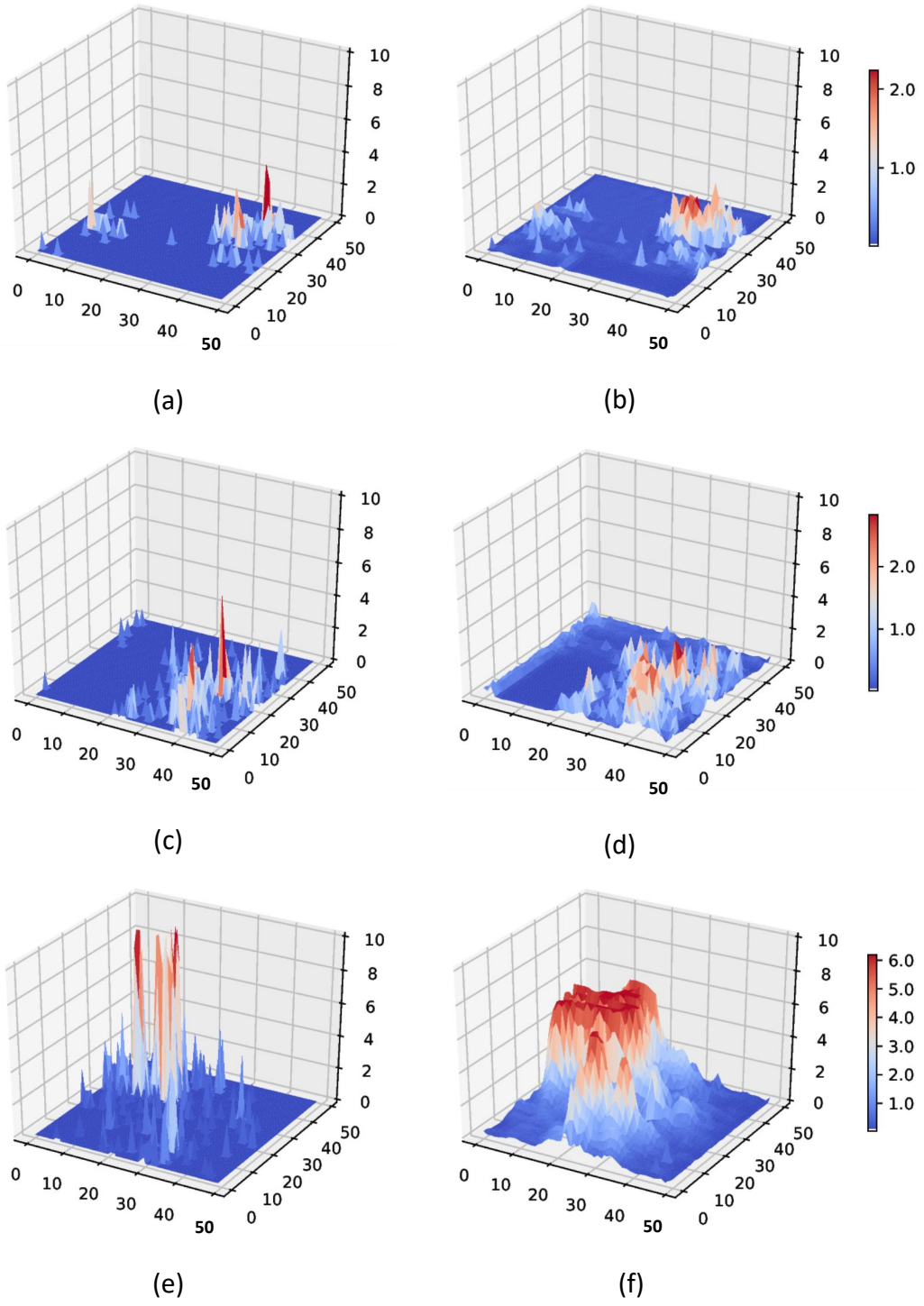


Figure 4.6: Snapshot of DRC heatmap prediction results: (a), (c) and (e) are DRC heatmap labels; (b), (d) and (e) are prediction results of our proposed J-Net Plus method.

4.6.3 Using Regression Results for Binary Classification

ROC curves of different binary classification methods are shown in Figure 4.7. Compared with the binary classifier method, the J-Net Plus method produces similar performance for label threshold $= 2$, and delivers superior performance for threshold 4 and 6. Another advantage of the J-Net Plus method over the binary classifier method is that, the regression model needs one-time training to give binary-classification results for various label thresholds; whereas one binary classifier is required to trained for one label threshold, consequently leading to several times longer training and inference overhead.

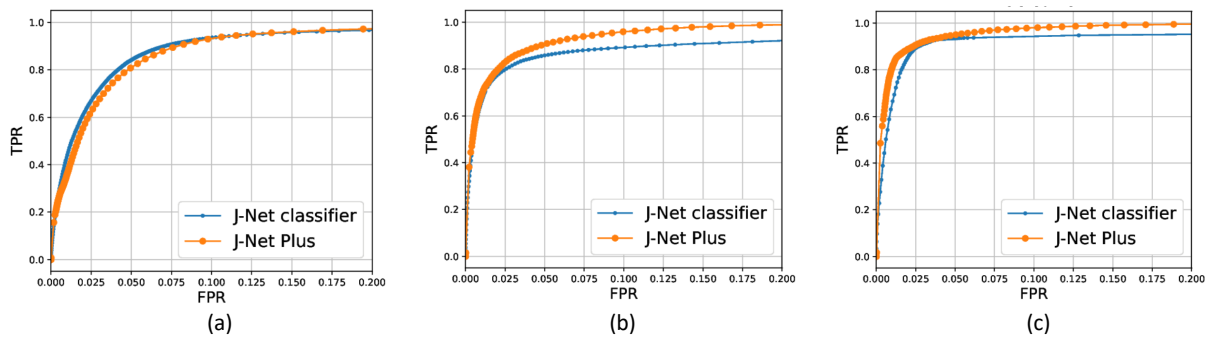


Figure 4.7: ROC curves for various binary-classification methods at different label thresholds: (a) th = 2, (b) th = 4, and (c) th = 6.

4.6.4 Training and Inference Time

It takes around 55 hrs to train our DRC heatmap predictor on a GeForce RTX 2080-Ti GPU. Because the trained model can be reused repeatedly for different and unseen designs of the same technology node, the amortized training cost is limited. As for feature extraction and inference time, it takes less than 1 minute for one layout design.

4.7 Conclusion

In this work, we present a DRC heatmap prediction method considering the non-determinism in routing. To tackle the noisy DRC labels caused by non-deterministic parallel routing, we com-

bine the strong modeling capability of deep neural network and the strength of stochastic models in coping with uncertainty. The stochastic distribution of DRVs in the layout is modelled by a Poisson point process and a Gaussian random field, based on which two novel techniques, the focal likelihood loss and the Gaussian random field layer, are proposed. Experiments results on industrial designs demonstrate that, these two techniques help achieve superior DRC heatmap prediction performance with noisy label data. Given that the issue of noisy label data caused by non-deterministic parallel computing is a general problem in applying machine-learning techniques to EDA tasks, we hope our solution can be somehow instructive for addressing this issue.

5. ROUTING-FREE CROSSTALK PREDICTION *

5.1 Introduction

Crosstalk-induced noise and delay variation have emerged as a serious concern in advanced technology nodes. Through capacitive coupling, a signal switching of one net causes crosstalk noise at its neighboring nets [90]. Noise amplitude can even reach up to 30% of V_{DD} [91], which may exceed the threshold voltage of transistors and lead to glitches, which impose a risk of logic errors and unwanted switching power consumption. Moreover, the coupling capacitance itself serves as extra load increasing both signal delay and power dissipation; the incremental delay due to coupling capacitance along a timing path can reach $300ps$ [92], comparable to clock periods of modern high-performance processors.

Nontheless, an accurate estimation of crosstalk effects is only possible after exact routing topology is available, i.e., after detailed routing. As detailed routing is among the last few steps in physical design, few rooms may remain to fix all the crosstalk-induced design problems even if we identify every crosstalk issue [90]. More flexibility for design changes can be found at global routing stage [93, 94]. Estimating crosstalk, however, becomes much harder at this point as exact routing information is not determined yet. There may also be still insufficient opportunities to fix significant crosstalk-induced problems even at global routing stage.

In this regard, many research efforts have been undertaken to estimate and mitigate crosstalk problems at earlier design stages, e.g., placement [92, 95]. There is usually a relatively large degree of design flexibility at earlier stages, which helps us resolve all the significant crosstalk issues if identified. The key problem of such an early-stage estimation, however, is the absence of routing information. It is mostly impossible to estimate crosstalk accurately with placement alone. Indeed, crosstalk driven placement [92, 95] resorts to global routing or trial routing for obtaining an approximate estimate of net adjacency information and thereby capacitive coupling among nets.

*

An evident drawback here is that global or trial routing is time-consuming and thus induces huge runtime costs.

5.1.1 Motivation

There is a tradeoff between accuracy of crosstalk prediction and design flexibility across three main relevant stages: placement, global routing, and detailed routing. An ideal crosstalk avoidance flow could be as follows.

1. During placement, significant crosstalk risks are identified and resolved by leveraging the relatively larger design flexibility.

2. Most of the remaining crosstalk risks are eliminated in global routing via layer assignment [96] and area routing [97].

3. In detailed routing, complete crosstalk avoidance is achieved with the precise crosstsalk evaluation.

While crosstalk avoidance at routing stages has been well studied [90, 93, 94, 97], the placement stage solutions [92, 95] are still far from being practical largely due to the dependence on trial/-global routing, which can easily take more than a half hour for a modern design. Indeed, pre-routing crosstalk prediction is a major weak link in realizing the ideal crosstalk avoidance flow.

One might consider employing existing routing congestion prediction methods [34] and use the results as a proxy of predicted crosstalk hotspots, since coupling capacitance correlates with routing congestion. Despite the correlation, Figure 5.1 shows that there exists difference between a routing congestion hotspot and a crosstalk-induced noise hotspot. One important reason for the mismatch is that noise is determined by not only coupling capacitance, but also electrical and logical parameters.

5.1.2 Contributions

In this chapter, we present a predictive method that can quickly identify a majority of crosstalk prone nets at placement stages, *without any routing information*. Such predictive identification

would not only assist crosstalk driven placement [92, 95], but also make other crosstalk mitigation techniques viable at placement stage, such as gate sizing [98] and buffer insertion [99]. Note that it is not necessary to identify all problematic nets as the goal of placement-stage techniques is to reduce crosstalk risk and make subsequent avoidance techniques feasible rather than completely solve all crosstalk problems. At the same time, the prediction must be very fast, at least an order of magnitude faster than global/trial routing.

We identify routing and net topology-related features, together with electrical and logical features, which affect crosstalk-induced noise and delay. Our approach leverages recent progresses on ML such as XGBoost [63]. In addition, graph-based ML techniques including GraphSage [100] and graph attention networks [101] are investigated for the crosstalk prediction. Given a cell placement solution for a design, the proposed approach predicts coupling capacitance, peak noise and crosstalk induced incremental delay for every signal net. Experimental results show that it can identify over 70% of nets with top crosstalk problems at false positive rate no greater than 2%. At the same time, its computation speed is two orders of magnitude faster than a conventional approach based on global routing. Detailed analysis of feature importance is also performed. To the best of our knowledge, this is the first approach to routing-free crosstalk prediction.

The rest of this chapter is organized as follows. The relation between our work and previous ones is described in Section 5.2 and the details of our proposed approach is presented in Section 5.3. Section 5.4 and 5.5 show our experimental setups and results, respectively. Section 5.6 concludes this chapter.

5.2 Related Work

Due to the importance of crosstalk avoidance, crosstalk estimation has been extensively studied in the past [91, 102–104]. Regardless accuracy and computation speed, all the previous methods require wire adjacency information, which is not available until detailed routing [90] or area routing [97]. Crosstalk estimation has also been studied targeting global routing [94, 105] and layer assignment stages [96]. Since no wire adjacency is available at these stages, the number of wire segments in a global routing cell is instead used to infer crosstalk in a probabilistic manner.

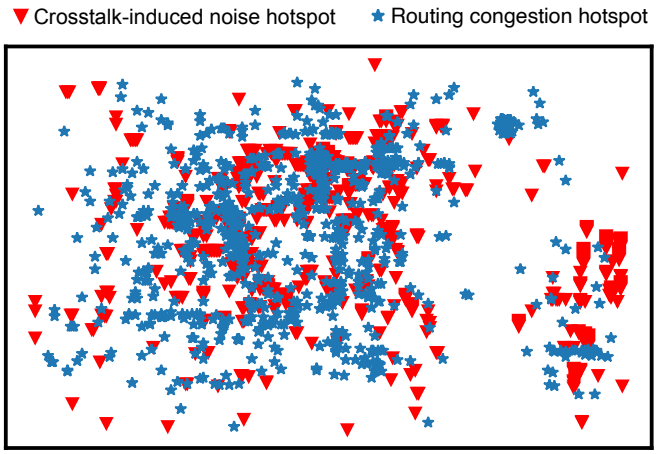


Figure 5.1: Routing congestion and crosstalk-induced noise hotspots (Reprinted with permission from [6]).

Crosstalk avoidance has also been addressed at placement [92, 95]. Such methods employ trial or global routing to get rough routing topology from placement. Based on the routing congestion information from the rough routing, coupling capacitance is estimated through curve-fitting from data of previously completed designs. Based on the estimated coupling capacitance and the routing routing topology, crosstalk noise is then computed using a simplified model, e.g., [92]. There have also been attemps to address crosstalk in even earlier design stages including technology mapping [106] and high-level synthesis [107]. Crosstalk avoidance is particularly investigated for bus design [108] where wire permutation is decided.

An ML-based crosstalk estimation is introduced in [109], to take crosstalk induced incremental delay into account within STA of routed designs. Instead of relying on the time-consuming signal integrity (SI) modes of STA tools, an ML model accounts for the effect of crosstalk during STA without turning on the SI mode. However, it can be only applied to routed designs. To the best of our knowledge, there is no previous work that can predict crosstalk without using any routing information. This is a void to be filled by our work for addressing crosstalk at placement and post-placement optimizations, such as gate sizing and buffer insertion.

5.3 Methodology

5.3.1 Overall flow

Crosstalk prediction by ML classification is a more attainable approach than ML regression. Moreover, classification results usually suffice for crosstalk avoidance in early design stages. We target at three classification tasks at the placement stage, to identify

1. The nets likely to have large coupling capacitance.
2. The nets likely to have large crosstalk-induced noise.
3. The nets likely to have long incremental delay due to crosstalk.

Figure 5.2 shows the Venn diagram of the above three sets of crosstalk-critical nets according to our experiment data. We can find that these sets have overlaps but they are not identical.

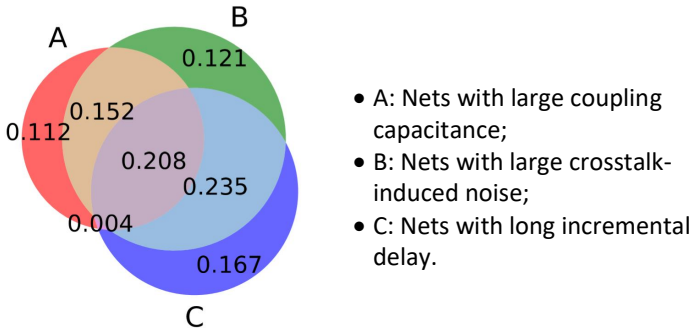


Figure 5.2: The Venn diagram of three crosstalk-critical net categories. The value on each region shows the proportion of the nets belonging to the region, normalized against the total number of crosstalk-critical nets (Reprinted with permission from [6]).

The crosstalk modeling flow is shown in Figure 5.3. Input features and ground truth information are extracted from the placement and post-routing databases, respectively, and they further constitute the ML database. By training and evaluating prediction performance of candidate ML models, e.g., XGboost, with the labeled data stored in the ML database, the most effective feature

sets and the best models for three crosstalk classification problems are determined, which can be used to fast identification of problematic nets in new placement instances.

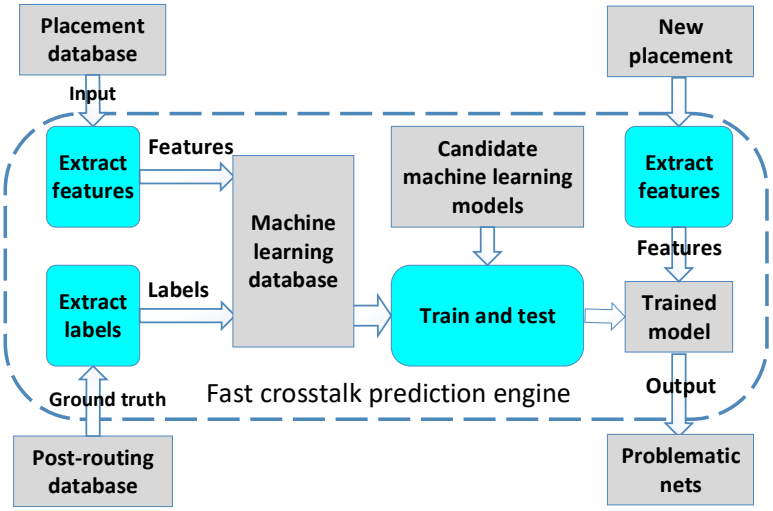


Figure 5.3: Crosstalk modeling flow (Reprinted with permission from [6]).

The raw input to crosstalk prediction engine includes placed DEF file, standard cell libraries, and the static timing analysis (STA) results generated after placement. The placed DEF contains the circuit netlist, and the locations of cells and input/output ports after placement. The standard cell library files are used to get the physical, electrical and logical properties of cells in the circuit. The STA results generated after placement give timing information such as cell delays, wire delays, and transition times. Net-based features are extracted from these files and then fed into ML models.

The ground truth information is extracted from the parasitic information file and the timing report in SI mode generated after detailed routing. For coupling capacitance prediction, the ground truth of a placement sample is represented by a binary vector of length n, where n is the total number of nets in a design. Each entry in the vector corresponds to a net, indicating whether its total coupling capacitance is larger than a given threshold. Crosstalk noise is measured at receiver pins. For each net, we calculate the peak noise amplitude and noise width product, denoted by AW, at the sink pins, because the noise susceptibility of digital logic gates is usually characterized

by a noise amplitude pulse width plot. The ground truth of a placement for noise prediction is denoted by a binary vector of length n. Similarly, we obtain a vector denoting whether the nets in a placement have crosstalk-induced incremental delays larger than a given threshold.

5.3.2 Feature Selection

5.3.2.1 Probabilistic Congestion Estimation

Routing congestion strongly correlates with crosstalk, since coupling capacitance tends to occur in congested areas. A probabilistic technique for congestion analysis is employed in our work due to its great runtime advantage over other techniques and its good correlation with the post-routing solution.

RUDY of a net is obtained via dividing the total wire volume going through the bounding box of the net by the bounding box area. We call two nets neighbors if their bounding boxes overlap. For the i-th net in a placement, with its neighboring nets denoted by $N(i)$, the total wire volume $\mathrm{wireVolume}(i)$ that go through its net-bounding-box is calculated as

$$\mathrm{wireVolume}(i) = \mathrm{HPWL}(i) + \sum_{j \in N(i)} \mathrm{HPWL}(j) \frac{\mathrm{overlapArea}(i,j)}{\mathrm{area}(j)}, \tag{5.1}$$

where $\mathrm{HPWL}(j)$ is the half perimeter wire length (HPWL) of net j, $\mathrm{area}(j)$ is the bounding box area of net j, and $\mathrm{overlapArea}(i,j)$ is the overlap area of the bounding boxes for nets i and j. The RUDY of the i-th net is then given by

$$\mathrm{RUDY}(i) = \frac{\mathrm{wireVolume}(i)}{\mathrm{area}(i)}. \tag{5.2}$$

Nets can be divided into long-range nets and short-range nets according to their HPWL[†]. In [34], it is shown that routing congestion has a stronger correlation with long-range nets than with shorter ones. In this regard, we also extract a feature, which is called **longRangeRUDY**, in a similar way as Equation (5.1) and Equation (5.2). The difference is that only long-range nets are

[†]We used $25\mu\mathrm{m}$ for the long-range net threshold in this work.

taken into account when computing the total wire volume.

5.3.2.2 Net Physical Information

Net physical information is necessary for crosstalk estimation. Because different routing topology of a net exposes it to different aggressors. Also, it leads to different electrical characteristics of interconnects, which have impacts on crosstalk noise and incremental delay. A few net physical features are extracted as follows.

- **HPWL** of the net-bounding-box.
- **area** of the net-bounding-box.
- **fan-out** of the source, i.e., the number of sinks of the net.
- **max-ss-distance**: the maximal distance between the source cell and the sink cells.

5.3.2.3 Product of the Wire-length and Congestion

As illustrated in [95], the total coupling capacitance of a net is proportional to the product of its wire-length and the unit coupling capacitance. We use the product of the HPWL and the RUDY/long-range-RUDY of a net as an indicator of its coupling capacitance.

- **HPWL-RUDY/HPWL-longRangeRUDY**: the product of HPWL and RUDY/longRangeRUDY of a net.

5.3.2.4 Electrical and Logical Features

The electrical properties of cells play an important role in crosstalk. For example, the crosstalk noise is affected by the source cell's resistance [92]. Also, the logic type of the source cell may affect the switching activity of the net, and consequently affect the noise and incremental delay. However, given the complicated timing models used by modern cell libraries, it is difficult to capture all crosstalk-related properties of a cell. To address this problem, we propose to use a logic-based encoding, along with the output capacitance, to represent a cell. First, library cells are partitioned into groups according to logic types. Each group contains cells with the same logic,

but can have different fan-in count (e.g., a two input NAND and a three input NAND belong to the same group) and different sizes. We use a variant of the one-hot encoding to encode the gate groups. Specifically, a vector of length N_g, the total number of gate groups, with only one non-zero entry is used to describe which group the cell belongs to. Unlike assigning 1 to the non-zero entry in the one-hot encoding, we assign the cell's fan-in to the non-zero entry. An additional feature, output capacitance, is used to capture the size of the gate. A few other electrical and logical structure features are also extracted.

- l_0 to l_{N_g} : the logic-based encoding of the source cell.
- **sourceCap**: output capacitance of the source cell.
- **sinkCap**: the sum of the input capacitance of sink cells.
- **fan-in**: the fan-in of the source cell.

5.3.2.5 Timing Information

The pre-routing timing report gives a rough estimation of wire delays and slews, which is informative for crosstalk prediction. For example, smaller slew often means stronger drive strength, consequently more resistant to aggregator's effect in term of incremental delay.

- **wireDelay** : the longest wire delay from the source to the sinks. Note it is a rough prediction from the pre-routing STA.
- **outputSlew** of the source cell from the pre-routing STA.

We also tried clock period, toggle rate and other timing related features. But we have not observed improvement brought by these features in our experiments.

5.3.2.6 Neighboring Net Information

Crosstalk noise and incremental delay depend not only on coupling capacitance, but also on the coupling location [92] (near the source or sink cells) and the aggressors' drive strength. We

estimate the coupling location via the relative location of the net's bounding box and its neighbors' bounding boxes.

- **#Neighboring nets**: number of neighboring nets.
- **#Neighboring long-range-nets**: number of neighboring long-range-nets.
- **mean-, std-, max-overlapArea**: the average/standard deviation/maximum of overlap areas between the net's bounding-box and neighboring nets' bounding-boxes.
- **mean-dist-source-overlap**: the average distance between the source cell and the geometric centers of overlap regions.
- **weighted-dist**: the average distance between the source cell and the geometric centers of overlap regions, weighted by the area of each overlap region.
- **dist-source-maxOverlap**: the distance between the source cell and the geometric center of the largest overlap region.

We use the sourceCap and outputSlew to represent the drive strength of a net. If a net is surrounded by nets with strong drive strength, then its aggressors is likely to have strong drive strength. The following features are employed to capture neighboring nets' drive strength.

- **mean-, std-sourceCap**: the average/standard deviation of sourceCap of neighboring nets.
- **sourceCap-maxOverlap**: the sourceCap of the neighboring net that has the largest overlap with the target net.
- **mean-, std-outputSlew**: the average/standard deviation of outputSlew of neighboring nets.
- **outputSlew-maxOverlap**: the outputSlew of the neighboring net that has the largest overlap with the target net.

Note that we do not use the electrical and logical features, and timing information in coupling capacitance prediction, since coupling capacitance is mainly determined by layout information.

5.3.3 Machine Learning-Based Crosstalk Estimation Models

Four popular ML techniques, logistic regression, neural network, random forest and XGboost, were used to model the mappings from the extracted features to coupling capacitance, crosstalk-induced noise and incremental delay. For each technique, three independent models were trained and fine-tuned for the three classification tasks. Implementation details about the models are shown as follows.

- **Logistic regression (LR)**: It is a linear classification model. We used scikit-learn [110] for the implementation, employing L_2 regularization to mitigate overfitting.

- **Neural network (NN)**: Multilayer perceptron neural-networks, implemented via the Pytorch [111], were employed. We implemented a narrow network with 3 layers for coupling capacitance prediction, and used deeper networks with 5 layers for crosstalk noise prediction and incremental delay prediction, since the mappings from input features to crosstalk noise and to incremental delay are considered to be more complex than the mapping from input to coupling capacitance. We also fine-tuned learning rate and other hyper-parameters to achieve the best performance.

- **Random forest (RF)**: A random forest consists of independently-trained decision trees, and the classification results are obtained by averaging the decisions of all trees or by voting [112]. A larger number of trees and deeper trees can make the model more expressive, but may result in overfitting; if a forest is too small and narrow, it may fall into underfitting. We tuned these parameters carefully to achieve balance between overfitting and underfitting. In our experiment, there were 100 trees with the maximum depth of 5 in the random forest implementation.

- **XGboost (XG)**: We employed the Gradient Boosted Regression Trees (GBRT) available in XGboost [63]. In GBRT, model trees are trained sequentially, where each training depends on the errors of previous ones. The final decision is the sum of decisions of all trees. In our experiment, the best results were obtained by 25 trees with the maximum depth of 8.

We also investigated the effectiveness of graph-based ML techniques for the crosstalk predic-

tion. Two net-centric graphs were generated: A dual circuit graph (DCG) and a proximity graph (PG). In a DCG, each node v_i represents a net, and a directed edge exists between a pair of nets (v_i, v_j) if a sink of v_i is the source of v_j. It can capture electrical connection information. A PG is used to capture the physical proximity among nets in the layout. Each node represents a net, and a unidirectional edge is created between two nets if the overlap area of their bounding boxes are larger than a given threshold[‡]. The weight of an edge is the area of the overlap region. Based on these two graphs, three ML models were implemented.

- **GraphSAGE with NN (on DCG)**: GraphSAGE [100] is a popular convolutional operation on graphs, which can learn a representation for each node integrating the information of the node itself and from its neighborhood. We used GraphSAGE operations on a DCG to integrate the information among electrically-connected nets, which is useful for crosstalk prediction. In particular, two GraphSAGE layers, implemented by Pytorch Geometric [113], were used to propagate electrical and logical features on the DCG, and output a learned feature vector for each net. Then the feature vector was concatenated with other features proposed in Figure 5.3.2 and fed to a NN for final classification.

- **GraphSAGE with XG (on DCG)**: It is the same as the above, except that the final classification is done with an XGBoost model instead of a NN.

- **NN with GraphAttention (on PG)**: GraphAttention [101] is an operation on graphs, suitable to capture dependency of neighboring nodes. For crosstalk prediction, there may be dependency among neighboring nets, since crosstalk occurs between physically-close nets. We use GraphAttention operations to capture such dependency on a PG. Firstly a multilayer perceptron neural network was applied to each net to output a rough classification result. Then two GraphAttention layers, implemented using Pytorch Geometric, were used to propagate rough classification result on the PG to get the final classification result.

[‡]We used $15\mu\mathrm{m}^2$ for the threshold in this work.

5.4 Design of Experiments

Our experiments were conducted on 12 designs from the IWLS 2005 benchmarks [114]. We generated a total of 304 placements with over 3 million labeled nets, by running a commercial design flow with various yet realistic parameter settings. Logic synthesis was performed using Synopsys Design Compiler and physical design was conduced with Cadence Innovus. We used ASAP7 standard cell library [115], an open-source 7-nm library. The clock period varied from 250ps to 400ps. The average number of nets and registers after placement as well as runtime of global routing (with the globalRoute command in Innovus) for each design are listed in Table 5.1. After detailed routing and RC extraction, STA was performed by Synopsys PrimeTime-SI and the analysis results were used as ground truth. The synthesis flow was conducted on a Linux server equipped with an Intel Xeon E5-2680 CPU.

The ground truth of coupling capacitance, crosstalk-induced noise and incremental delay are real numbers. They were transformed into binary classification labels via pre-defined thresholds (called *ground truth thresholds*). The thresholds used for coupling capacitance, noise and incremental delay classifications were 0.5fF, $2V\,\text{ps}$ and 2ps, respectively[§].

5.4.1 Model Training and Testing

5.4.1.1 Training & Testing Schemes

For the labeled data that we can obtain, two different partition schemes between training and testing sets were applied for the model evaluation.

- **Scheme 1:** Testing on nets from unseen placement instances. For each design, we randomly chose three placement solutions and used their labeled net samples as testing data, while nets from other solutions were used as training data. Please note testing placement instances may involve some same designs as those in the training set, although their placements were different.

[§]Coupling capacitance of 0.5fF is about 60% of the input capacitance of a buffer in the ASAP7 library. Also, 2ps is about 1% of the clock period. In Section 5.5.5, we investigate how classification performance changes with the ground truth thresholds.

Table 5.1: Benchmarks used in our experiment. The groups are for training/testing data partitioning (Section 5.4.1.1) (Reprinted with permission from [6]).

Design	#Nets	#Registers	GR Runtime	Group ID
systemcdes	3590	190	3s	1
tv80	4950	353	9s	2
systemcaes	5308	685	13s	3
mem_ctrl	6473	936	8s	4
wb_dma	6936	486	5s	4
ac97_ctrl	8397	1820	9s	3
usbf_funct	8629	132	13s	2
fpu	14416	542	41s	1
pci	19318	174	20s	1
aes_core	37105	527	21s	2
ethernet	47051	10002	109s	3
vga_lcd	74467	16038	203s	4

- **Scheme 2:** Testing on nets from unseen designs. The 12 designs were divided into four groups according to their numbers of nets, as shown in Table 5.1, making the four groups have similar amount of net samples. Four-fold cross-validation was performed. To be specific, four rounds of experiments were ran and their results were averaged. In each round, all net samples from three groups were taken as training data, while nets from the only remaining group were used as testing data. Each round had distinct testing designs. This is a stricter testing scheme than scheme 1 as not only the testing placement instances but also the testing designs are completely unseen during training.

5.4.1.2 Baseline Method

A global routing-based method similar to [92] was implemented as the baseline method. We ran global routing to get congestion of each global routing cell (gcell). Then, we generated coupling capacitance of each gcell $C(i,j)$ via curve-fitting as in [92]. The total coupling capacitance of each net CC was estimated to be $CC = \sum_{i,j} C(i,j)$, where (i,j) is a gcell traversed by the net. The noise of each net N was estimated as $N = \sum_{i,j} R_1(i,j)C(i,j)$, where $R_1(i,j) = a_1 d(i,j) + b_1$ and $d(i,j)$ is the Manhattan distance from the driver to (i,j). Similarly, the incremental delay was estimated as $D = \sum_{i,j} R_2(i,j)C(i,j)$, where $R_2(i,j) = a_2 d(i,j) + b_2$. The coefficients a_1, b_1, a_2

and b_2 were obtained via curve-fitting. The CC, N and D were then used to identify problematic nets.

5.4.2 Performance Metrics

The proportion of crosstalk-critical nets to the total nets in a design is typically small. The positive samples (i.e., crosstalk-critical nets) in our classification problems is hence much less than the negative samples in general. In this regard, we evaluated prediction performance with the following four metrics widely used for imbalanced data set [116], i.e., FPR, TPR, F1-score and Balanced accuracy: $\mathrm{BAcc} = \frac{(1-\mathrm{FPR})+\mathrm{TPR}}{2}$.

The raw output of our crosstalk prediction engine is a scalar in $[0, 1]$ for each net, which can be viewed as the probability of the net being crosstalk-critical in terms of coupling capacitance, noise or incremental delay. A classification threshold is required to turn the raw output into binary prediction label. A high threshold will lead to high TPR, but also high FPR; and vice versa. We plotted the ROC curves (introduced in Section 3.6) of prediction results, which describe the trade-off between TPR and FPR by varying the classification thresholds. The AUC of ROC curves (introduced in Section 3.6) was also calculated.

5.5 Results

5.5.1 Crosstalk Prediction: Scheme 1

The classification results from Scheme 1 are shown in Tables 5.2, 5.3, and 5.4. Overall, XGboost models achieved the best performance, even beating the baseline method in all the three classification tasks, in terms of most of the performance metrics we used. The random forest models achieved the second best results.

For coupling capacitance prediction, the four modeling techniques gave similar results. It means that there is a strong correlation between the extracted features and the coupling capacitance of nets. Thus, even the linear classification model, logistic regression, can capture such correlation very well. However, the performance of logistic regression was obviously inferior to non-linear techniques in the noise and incremental delay prediction, especially in the latter. It suggests the

Table 5.2: Coupling capacitance prediction results (Scheme 1) (Reprinted with permission from [6]).

Model	FPR	TPR	F1	BAcc	AUC
Baseline	1.10%	79.49%	52.67%	89.20%	0.993
LR	1.04%	81.98%	47.64%	90.47%	0.905
NN	1.01%	81.65%	48.24%	90.32%	0.994
RF	1.01%	82.45%	48.48%	90.72%	0.993
XG	0.94%	83.17%	50.58%	91.12%	0.994

Table 5.3: Crosstalk noise prediction results (Scheme 1) (Reprinted with permission from [6]).

Model	FPR	TPR	F1	BAcc	AUC
Baseline	1.30%	50.45%	40.19%	74.58%	0.974
LR	1.38%	68.99%	43.40%	83.81%	0.838
NN	1.23%	67.51%	44.85%	83.14%	0.987
RF	1.16%	68.82%	46.77%	83.83%	0.987
XG	0.99%	70.25%	50.65%	84.63%	0.990

Table 5.4: Incremental delay prediction results (Scheme 1) (Reprinted with permission from [6]).

Model	FPR	TPR	F1	BAcc	AUC
Baseline	4.04%	57.18%	21.18%	76.57%	0.947
LR	2.08%	61.44%	30.70%	79.68%	0.797
NN	2.08%	70.85%	34.58%	84.38%	0.980
RF	2.03%	72.78%	35.80%	85.37%	0.978
XG	1.88%	77.01%	39.17%	87.56%	0.986

mappings from extracted features to crosstalk-induced noise and incremental delay are largely non-linear, which can not be captured well by linear models. One other observation is that ensemble tree-based modeling, i.e., random forest and XGboost, outperformed the neural network approach in our tasks.

Figure 5.4 shows the trade-off between TPR and FPR in XGboost models. It can be seen that the coupling capacitance, crosstalk-induced noise and incremental delay predictions are progressively more difficult.

5.5.2 Crosstalk Prediction: Scheme 2

Results from Scheme 2 are similar to those from Scheme 1. Among the four modeling techniques, XGboost achieved the best performance in all three classification tasks. Detailed results for coupling capacitance, noise and incremental delay predictionns are listed in Tables 5.5, 5.6

Table 5.5: Coupling capacitance prediction results (Scheme 2) (Reprinted with permission from [6]).

Model	FPR	TPR	F1	BAcc	AUC
Baseline	1.06%	78.93%	53.14%	88.94%	0.993
LR	1.03%	80.91%	50.98%	89.94%	0.899
NN	0.98%	80.73%	51.90%	89.87%	0.959
RF	0.90%	81.62%	54.16%	90.36%	0.994
XG	0.89%	81.87%	54.62%	90.49%	0.994

Table 5.6: Crosstalk noise prediction result (Scheme 2) (Reprinted with permission from [6]).

Model	FPR	TPR	F1	BAcc	AUC
Baseline	1.56%	50.88%	37.47%	74.66%	0.970
LR	1.62%	75.15%	47.17%	86.77%	0.868
NN	1.45%	66.56%	45.09%	82.55%	0.846
RF	1.31%	76.34%	52.27%	87.51%	0.987
XG	1.28%	78.37%	53.69%	88.54%	0.990

Table 5.7: Incremental delay prediction results (Scheme 2) (Reprinted with permission from [6]).

Model	FPR	TPR	F1	BAcc	AUC
Baseline	4.35%	52.83%	18.71%	74.24%	0.937
LR	2.43%	69.29%	33.19%	83.43%	0.834
NN	2.39%	73.10%	35.07%	85.35%	0.939
RF	2.10%	74.74%	38.45%	86.32%	0.978
XG	1.80%	76.25%	42.35%	87.23%	0.986

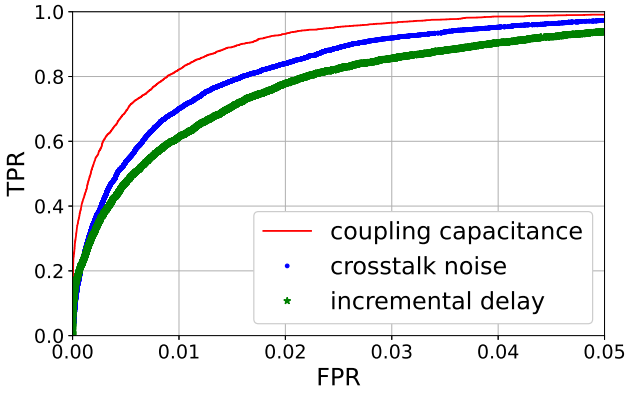


Figure 5.4: ROC curves of the XGboost prediction results (Reprinted with permission from [6]).

and 5.7, respectively. Note that Scheme 2 is stricter than Scheme 1 in that not only the testing placement solutions but also the testing designs are completely unseen during training. But the prediction results from Scheme 2 were similar to those from Scheme 1, meaning the proposed modeling generalized well on unseen designs.

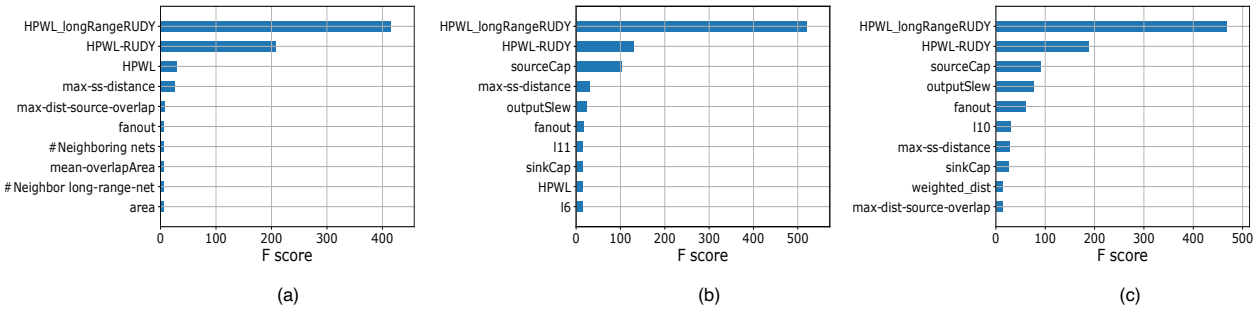


Figure 5.5: Top-10 important features: (a) in coupling capacitance, (b) in crosstalk noise, and (c) in incremental delay predictions (Reprinted with permission from [6]).

5.5.3 Crosstalk Prediction: Graph-Based Models

For coupling capacitance prediction, we did not adopt "GraphSAGE + NN" and "GraphSAGE + XG" models, which act on the DCG, since coupling capacitance does not depend on electrical connection. The TPR of the "NN + GraphAttention" model was approximately 1% higher than that of the NN model, at the same FPR, but still inferior to the XGboost model.

Tables 5.8 and 5.9 show the results of graph-based models for crosstalk-induced noise and incremental delay prediction using Scheme 1. It seems that graph-based learning techniques did not help improve prediction performance. The reason might be that electrical connection information is not significant for crosstalk prediction at early stages, so the DCG is not helpful; physical proximity information can be well captured by extracted features, such as RUDY, so it might not be necessary to use the PG. Among the graph-based techniques the GraphSAGE + XGboost model achieved the best result, similar to that of the XGboost method. The results of these two approaches were also fairly consistent. In particular, 94% and 95% of the classified positive samples were identical in the two models, for noise and incremental delay predictions, respectively.

Table 5.8: Results of graph-based models with dropping some features for noise prediction (Scheme 1) (Reprinted with permission from [6]).

Model	FPR	TPR	F1	BAcc	AUC
XG	0.99%	70.25%	50.65%	84.63%	0.990
GS + NN	1.10%	63.27%	44.89%	81.09%	0.987
GS + XG	1.03%	70.91%	50.22%	84.94%	0.990
NN + GA	1.18%	65.89%	44.92%	82.36%	0.988
Layout only	1.26%	57.45%	39.10%	78.09%	0.981
No timing info.	1.01%	69.40%	49.86%	84.19%	0.989
Binary encoding	0.99%	69.78%	50.34%	84.39%	0.990

Table 5.9: Results of graph-based models with dropping some features for incremental delay prediction (Scheme 1) (Reprinted with permission from [6]).

Model	FPR	TPR	F1	BAcc	AUC
XG	1.88%	77.01%	39.17%	87.56%	0.986
GS + NN	2.09%	62.70%	31.15%	80.31%	0.822
GS + XG	1.95%	76.77%	38.23%	87.41%	0.984
NN + GA	1.94%	65.27%	33.65%	81.67%	0.861
Layout only	2.66%	61.50%	26.28%	79.42%	0.966
No timing info.	2.09%	73.45%	35.45%	85.68%	0.983
Binary encoding	1.90%	76.39%	38.74%	87.25%	0.986

5.5.4 Feature Importance Analysis

One important advantage of tree-based models over neural network models is that they are more interpretable. After training a tree-based model, users can check which features are most important in building the decision trees. Importance can be defined from various aspects. One commonly-used metric is "Gain", which is the improvement in accuracy brought by a feature. importance shows the top-10 important features in coupling capacitance, crosstalk-induced noise and incremental delay predictions, in term of "Gain".

It can be seen that the layout features played an important role in the three crosstalk prediction because crosstalk heavily depends on layout. It is noteworthy that HPWL-longRange-RUDY was even more important than HPWL-RUDY. It seems that crosstalk is more relevant to long-range-nets than to short-range-nets. As for crosstalk-induced noise and incremental delay prediction, we can see that electrical features (e.g., sourceCap and sinkCap), logical features (e.g., logic-based

encoding for the source cell: l_6, l_{10} and l_{11}), the timing information (e.g., outputSlew) and the neighboring net information (e.g., the weighted-dist and the max-dist-source-overlap) also have great importance. To better show their importance, we conducted additional experiments where only layout features were used. From the last three rows of Table 5.8 and Table 5.9, we can find that prediction accuracy dropped significantly without electrical and logical features as well as timing information. Moreover, we explored the effects of dropping timing information alone, i.e., the wire delay and slew-related features. It is interesting that dropping these features does not hurt the noise prediction, but does hurt incremental delay prediction noticeably. Also, we found a small decrease in prediction performance if we use the conventional binary encoding of cells instead of the proposed encoding.

5.5.5 Impact of Ground Truth Threshold and Training Sample Counts

We investigated how F1-score and AUC of ROC change with the ground truth thresholds using XGboost. In Figure 5.6, we can find that a higher ground truth threshold led to a larger AUC of ROC. For the prediction of crosstalk-induced noise and incremental delay, however, the $0.5\times$ threshold resulted in the highest F1-score.

If we train one classification model with the original threshold and train another one with the $2\times$ threshold, their prediction results might not be consistent, in the sense that a net might be predicted larger than the $2\times$ threshold by one model but predicted smaller than the $1.0\times$ threshold by the other. We also investigated the consistency of the XGboost model. From our experimental data we can find that, for coupling capacitance, all the nets predicted positive with the $2\times$ threshold by the XGboost were also predicted positive with the original threshold. In terms of crosstalk-induced noise and delay, among all the nets predicted larger than the $2\times$ threshold, 99.77% and 97.26% were also predicted larger than the original threshold, respectively. This indicates that the XGboost model is stable and consistent.

The number of training samples also has a significant impact on prediction performance. Figure 5.7 shows how the TPR of XGboost models changes with the number of training samples, at the same FPR. We can observe an about 4% drop in TPR if we only use one third of training

samples.

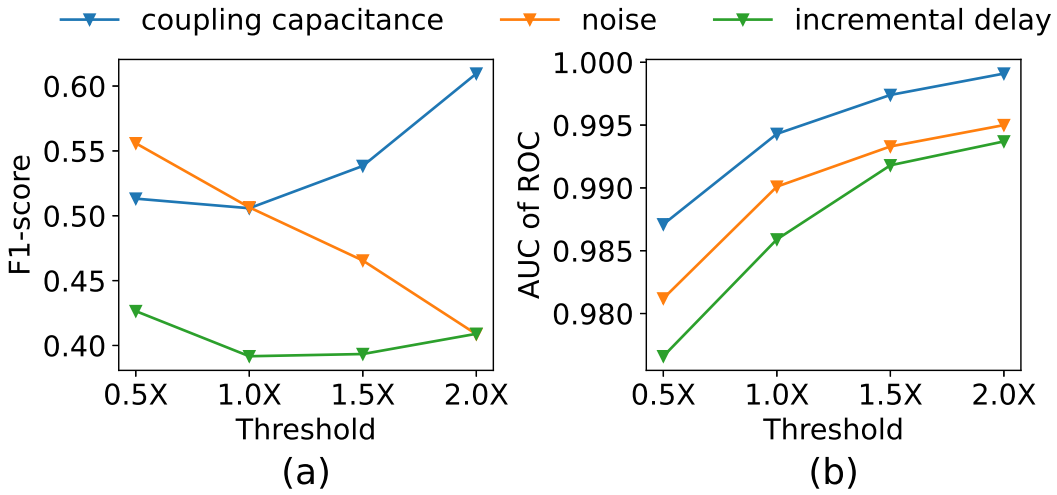


Figure 5.6: Impact of ground truth threshold in XGboost. (a) F1-score, and (b) AUC of ROC (Reprinted with permission from [6]).

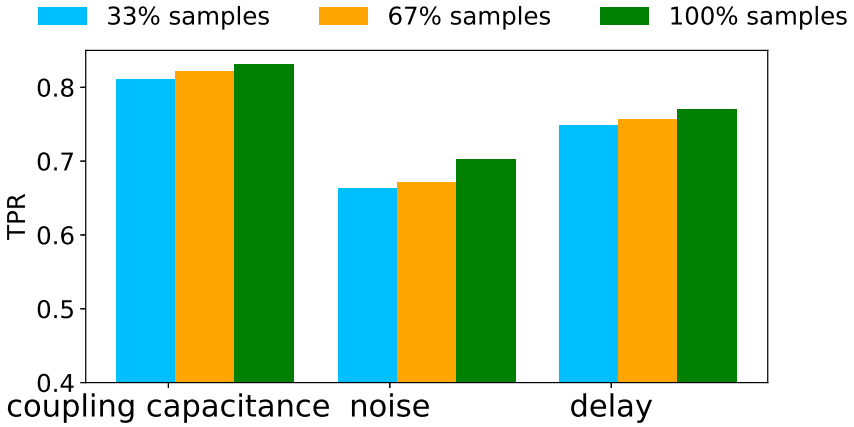


Figure 5.7: Impact of training sample counts in XGboost (Reprinted with permission from [6]).

5.5.6 Runtime and Memory Usage

Given the extracted features illustrated in Section 5.3.2, the inference time of the three XGboost models on an Intel XeonE5-2680 CPU only took $0.003s$–$0.006s$ for one placement instance of

various sizes in our benchmarks. Compared to the global routing runtime in Table 5.1, the XGboost models can achieve over $500\times$ speedup. For large industrial designs, the runtime advantage of the proposed method will be more outstanding. Besides, the training of the three XGboost models takes only about $10s$ on a RTX 2080Ti GPU. Moreover, a XGboost model consumes less than 300KB memory. And the peak runtime memory usage of XGboost-based prediction is about 1GB. As such, the proposed crosstalk prediction approach is very runtime- and memory-efficient.

5.6 Conclusion

In this chapter, we present a routing-free ML-based crosstalk prediction framework. Given a placement, we extract net physical information, along with electrical, logical, and timing-related features. Machine-learning techniques are then employed to train crosstalk prediction models, which classify the nets that are likely to have large coupling capacitance, crosstalk-induced noise, or incremental delay. Experimental validation on 12 benchmark circuits shows that the proposed method can classify more than 70% of crosstalk-critical nets after placement with a FPR of less than 2%. The computation speed is two orders of magnitude faster than a conventional method based on global routing. These results demonstrate that the proposed framework can serve as an accurate early-stage crosstalk evaluation engine that does not require routing information.

6. FLOWTUNER: A MULTI-OBJECTIVE MULTI-STAGE EDA FLOW TUNER EXPLOITING PARAMETER KNOWLEDGE TRANSFER*

6.1 Introduction

As the VLSI technology advancement continues in recent years, the design productivity gap has been ever-growing in even faster paces as pointed out by the ITRS prediction [11]. Accordingly, the capabilities of EDA tools and flows have evolved with a myriad of parameters and options to cope with the increased complexity and scale of designs. Ironically, the expanded design solution space due to these parameters and options often requires experienced designers' intuitions and knowledge to utilize full potentials of EDA tools. In high performance microprocessor designs, it is not uncommon to have significantly different PPA results on the same design with different settings of parameters [117].

In order to navigate the solution space in a more systematic and intelligent manner, automatic flow parameter tuning has been extensively studied in recent years [24, 56–58, 117–124]. Various sampling methods for tool parameter tuning have been studied, e.g., machine learning-based models [58, 123], Gaussian process [119], or bandit method [56, 120]. The shared goal is to efficiently find an optimal parameter configuration for a given design. Some works utilize knowledge provided by designers [117, 122] to enhance tuning efficiency. In [58], a feature-importance sampling and tree-based parameter tuning, named FIST, is proposed. It learns parameter importance automatically from legacy designs, which is utilized in the sampling for new designs. In [57], a latent factor model is trained with archived design data to recommend the best parameter configurations of a new design. A reinforcement learning-based method is proposed in [124] to suggest the best parameter configuration of a new design in one shot. The authors of [120] address excessive runtime of flow tuning by parallel computing. Lamda [121] accelerates FPGA flow tuning by

*©2021 IEEE/ACM. Reprinted, with permission, from Rongjian Liang, Jinwook Jung, Hua Xiang, Lakshmi Reddy, Alexey Lvov, Jiang Hu, Gi-Joon Nam, "FlowTuner: A Multi-Stage EDA Flow Tuner Exploiting Parameter Knowledge Transfer", Proceedings of the 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD).

estimating post-routing PPA of candidate configurations after the technology mapping & packing stage and halting low-ranking ones.

Although there has been a series of studies on EDA tool parameter tuning problem, there exist a few major limitations in previous works. First, previous works either target at a single stage of the EDA flow [118, 124], thus missing significant opportunities of cross-stage optimization, or tune flow parameters of multiple stages as a whole [24, 57, 58, 121], thereby coping with an extraordinarily huge search space. Second, though a few works [120, 121] address the tuning runtime, the excessive runtime of EDA flow calls for more systematic and effective flow control schemes. And last but not least, most of previous techniques focus on search efficiency on given designs [56, 119, 120, 123], without exploiting knowledge transfer across designs. In [58], transferred knowledge is limited to parameter importance. Exploration on new designs is under-emphasized in [57, 124], and their performances heavily rely on the accuracy of underlying machine learning models trained with archived design data, which is risky due to complexity of EDA flows and diversity of designs. Third, the design flow tuning is a multi-objective optimization as hinted by PPA metrics. Yet, many prior arts [57, 117, 119] cast it as a single-trade-off problem by employing a weighted sum of metrics. These weight terms require another trade-off analysis effort to produce desirable results. A truly multi-objective tuning should be able to provide the insights among competing metrics and be flexible to provide desirable balance points.

6.1.1 Contributions

We present a multi-stage EDA flow parameter tuner, named *FlowTuner*, to address the aforementioned limitations. Specifically, FlowTuner enables effective and efficient design flow tuning capability by utilizing both *exploitation* using transferred knowledge from archival design data and *exploration* via a multi-stage cooperative co-evolutionary framework [125]. For efficient computation resource management, the notions of *jump-start* and *early-stop* are actively deployed. The main contributions of this work are:

- A Cooperative Co-evolutionary (CC) framework for efficient *exploration* in the multi-stage design solution space (Section 6.3). Cooperative co-evolution is inspired by *mutualism*, an

ecological phenomenon, where mutually beneficial species evolve together to achieve the best common benefits. We map the flow parameters for each stage to *species*, the tuning process to *evolution*, and the final Quality-of-Result (QoR) to *common benefits*. For individual stage tuning, a search engine based on ant colony optimization (ACO) is developed, while the tuning of multiple design stages are orchestrated in an interactive and cooperative manner to achieve the best final QoR.

- The deployment of novel *jump-start* and *early-stop* capabilities for runtime efficiency (Section 6.4). *Jump-start* actively exploits reusable design outcomes. *Early-stop* employs a branch-and-bound strategy integrated with a probabilistic reward upper bound prediction technique to systematically prune out non-promising parameter configurations prematurely.

- Automatic knowledge extraction from archival data (*exploitation*), and its integration with the *exploration* search engine for tuning efficacy (Section 6.5). Three types of parameter knowledge: importance, bias and interaction are automatically extracted and utilized for the warm-up of the search engine.

- The enablement of efficient Pareto front exploration with concurrent multi-trade-off learning capability (sec:MO).

- Demonstration that superior solutions are obtained in 50% less turnaround compared against prior techniques (Section 6.6).

The remainder of this chapter is organized as follows. Section 6.2 gives an overview of the proposed FlowTuner. Details of the cooperative co-evolutionary search engine are introduced in Section 6.3. Section 6.4 describes the details of flow jump-start and early-stop techniques. Our knowledge transfer approach is elaborated in Section 6.5. Single-trade-off tuning results are shown in Section 6.6. Our proposed Pareto front exploration approach via concurrent multi-trade-off learning and the evaluation results are shown in Section 6.7. Section 6.8 gives some concluding remarks.

6.2 FlowTuner Overview

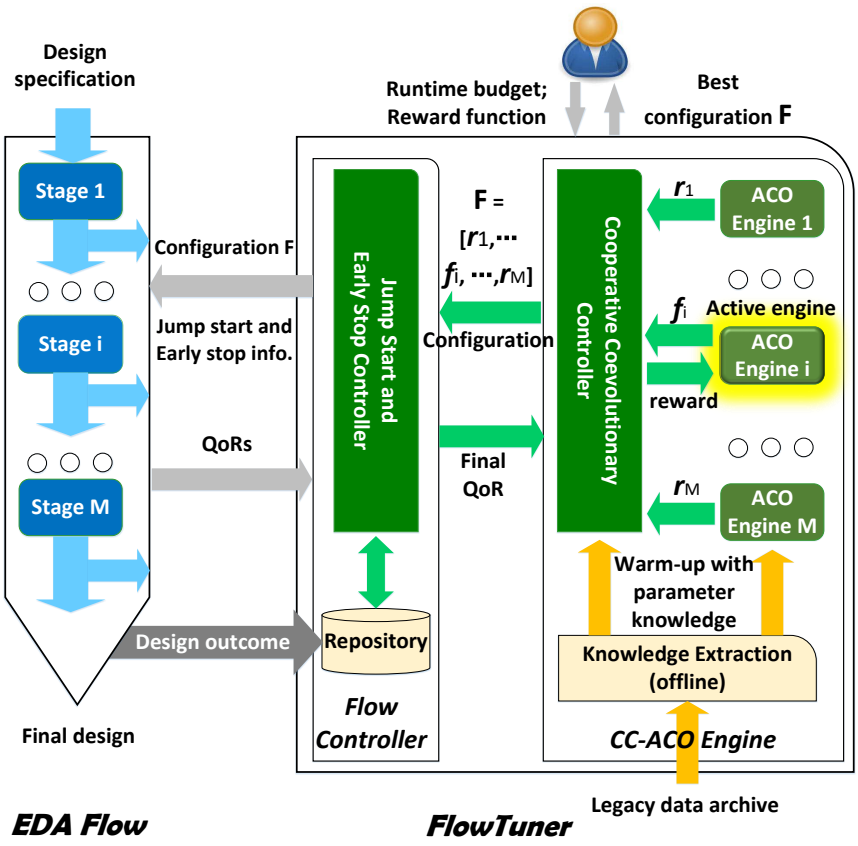


Figure 6.1: Overall architecture of FlowTuner (Reprinted with permission from [7]).

Figure 6.1 shows the overall architecture of FlowTuner. It consists of the CC-ACO engine and the Flow Controller. The CC-ACO engine contains a set of ACO engines, each of which is in charge of the tuning of the corresponding design stage. If the design flow consists of M design stages, FlowTuner instantiates M ACO engines. The i-th ACO engine generates the parameter configuration to try for the corresponding stage based on the reward values obtained so far. A co-operative co-evolutionary controller (CCC) orchestrates the behaviours of the ACO engines so that the limited compute resource budget is allocated well for effective parameter tuning. It also combines the parameter configuration of each stage suggested by individual ACO engines, and send the configuration to the Flow Controller. The CC-ACO engine is so named because it leverages

CC and ACO frameworks.

Given a full parameter configuration from CC-ACO, the Flow Controller obtains QoR information of individual stages by running the EDA flow. For efficient parameter tuning, the Flow Controller manages the EDA flow by maneuvering the *flow jump-start* technique, which skips previously-exercised parameter configurations, as well as the *flow early-stop* technique, which proactively halts hopeless trials. To this end, it keeps track of the design outcomes obtained during the online tuning in the repository. The final QoR information returned from the Flow Controller is then used to update the CCC and the individual ACO engines.

The efficiency of the *online* parameter tuning in FlowTuner can be greatly enhanced via offline warm-up. To be specific, the knowledge of flow parameters are extracted before the parameter tuning from the legacy design archives. The extracted knowledge are then employed to initialize the importance of individual stages in the CCC as well as the pheromone levels of each ACO engine. The extracted knowledge can be updated offline periodically once sufficient amount of new design data are accumulated. The goal of offline warm-up is to actively bias the behaviour of CC-ACO engine with the parameter knowledge transferred from legacy design data.

For the simplicity of presentation, we introduce the following notations and use them in the rest of this paper:

- $\boldsymbol{f}_i \in \mathbb{R}^{s_i}$: a partial parameter configuration for the i-th design stage having s_i tunable parameters. An element f_j of $\mathbf{f}_i$ represents a single parameter.
- $\mathbf{F} = [\boldsymbol{f}_1, \boldsymbol{f}_2, \ldots, \boldsymbol{f}_M]$: a complete parameter configuration of the design flow having M stages.
- $\boldsymbol{r}_i \in \mathbb{R}^{s_i}$: the best, *representative* parameter setting of the i-th design stage thus far.
- $r(*)$: a weighted sum of QoR metrics of interest, called *reward*.

6.3 Cooperative Co-Evolutionary Ant Colony Optimization Engine

As shown in Figure 6.1, CC-ACO engine has a hierarchical structure. The cooperative co-evolutionary controller (CCC) orchestrates the entire design flow while a set of ACO engines con-

trol corresponding design stages individually. The CCC coordinates the behaviors of the ACO engines by (1) adaptively allocating computation resources and (2) calling for cooperation when to evaluate new parameter configurations.

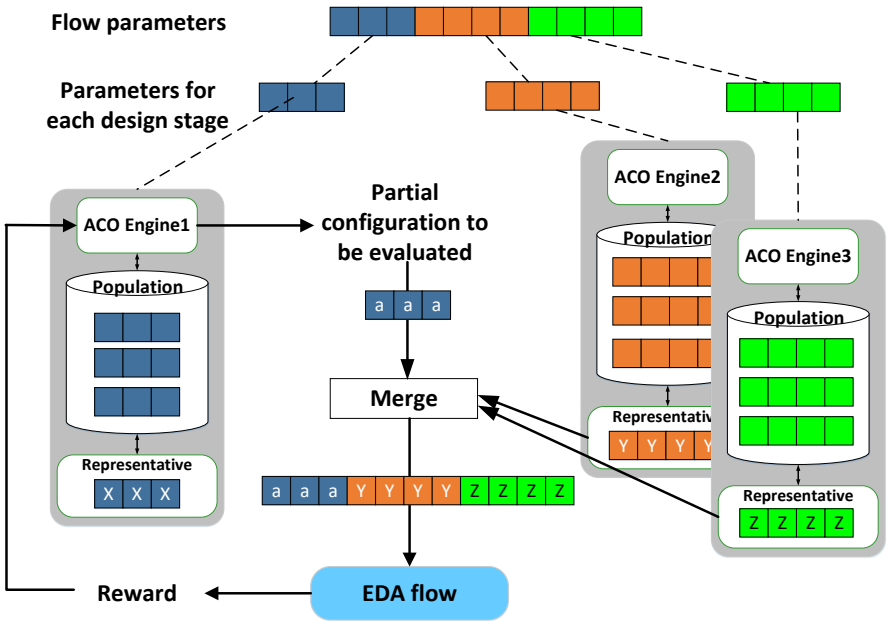


Figure 6.2: Evolving of solutions in the cooperative co-evolutionary framework (Reprinted with permission from [7]).

6.3.1 Cooperative Co-Evolutionary Controller (CCC)

As shown in Figure 6.2, the flow parameter tuning problem, a high-dimensional combinatorial optimization problem[†], is decomposed into a set of more tractable sub-problems: parameter tuning for each design stage. Each sub-problem is solved by an individual searching engine. However, the evaluation of a partial configuration suggested by one search engine calls for the cooperation of all engines. To be specific, a complete configuration is obtained by merging the partial configuration and the *representative* ("best-so-far") configurations for remaining design stages, and then the fitness of the partial configuration is evaluated in terms of the reward of the complete configuration in which it participates. In other words, the evaluation of a configuration for one stage considers

[†]Continuous parameters are discretized.

how well it cooperates with configurations in other stages, which coordinates the tuning for all design stages to achieve the best final QoR.

Algorithm 1 depicts the behavior of the CCC. The online tuning process can contain multiple epochs, each of which consists of dozens of trials. When a tuning epoch starts, the budget of #trials in this epoch is allocated to each ACO engine (Line 3) based on the stage importance, which reflects the potential QoR benefits brought by parameter tuning in a design stage[‡]. As shown in Figure 6.2, only one stage is activated for parameter tuning in each trial. The parameters for remaining stages are set to the *representatives*. Lines 7–14 show the process of generating a new configuration, then it is passed to the flow controller that launches a flow trial to obtain the final QoR of the entire flow (Line 15). Using this final QoR, the active ACO engine is updated to digest the knowledge contained in QoR (Lines 17–18). When the stage runtime budget is used up, the CCC moves on to the next stage.

Once the tuning of every stage in one epoch is completed, the *stage importance* is updated according to the QoR benefits brought by applying the parameter tuning in the stage. It is measured using the Fitness-based Area-Under-Curve (F-AUC) metric [8]. An example of F-AUC scores computation is given in Figure 6.3. The curves in Figure 6.3 are drawn according to a stage list $s\_list = [1, 1, 1, 2, 2]$ and a reward list $r\_list = [0, 2, 1, 3, 0]$, where the first entry in s_list being 1 and the first entry in r_list being 0 mean that allocating one trial to stage 1 leads to a reward of 0. Sorting r_list gives $(3, 2, 1, [0, 0])$, where $[0, 0]$ stands for two trials with the same reward. Replacing each reward by the index of the corresponding stage gives $(2, 1, 1, [1, 2])$, based on which a fitness curve can be drawn for each stage, e.g., the first entry 2 results in segments from points (0,0) to (1,0) in (a) and from (0,0) to (0,1) in (b); '[1,2]' leads to the diagonal ones from (1,2) to (2,3) in (a) and from (2,1) to (3,2) in (b). The F-AUC rewards for assigning computation resource to stage 1 and stage 2 can be calculated as the shaded areas in (a) and (b), respectively; $\boldsymbol{u} = [\frac{2.5}{2.5+3.5}, \frac{3.5}{2.5+3.5}]$.

The F-AUC is originally designed to assess the empirical quality of strategies by comparing

[‡]The initial stage importance for each stage is assigned based on the knowledge extracted from archival data, as we describe in Section 6.5.

the final rewards after their applications. In FlowTuner, a strategy corresponds to the computation resource assignment to a stage. The F-AUC metric has a few nice properties suitable for our application. (1) It is a comparison-based metric, i.e., the F-AUC scores depend on the relative ranking rather than the actual reward values. Hence, our F-AUC-based stage importance measure works well with various reward functions, though they might lead to rewards with highly-different magnitudes. (2) Computation resource assigned to different stages can have a big difference. F-AUC metric is fair in the sense that it does not favor stages with large numbers of trials or the ones with small numbers of trials, making it suitable for stage importance evaluation in the cooperative co-evolutionary framework. The principle of stage importance-driven resource allocation (Lines 3) is to invest more computation resources to stages that have higher potential reward benefits (i.e., better QoR).

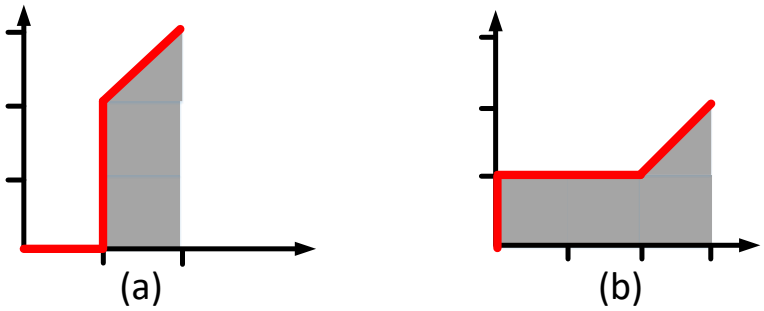


Figure 6.3: Computation of F-AUC scores (adapted from [8]). (a) Fitness curve associated with stage 1; (b) Fitness curve associated with stage 2. These curves are drawn according to a stage list $s\_list = [1, 1, 1, 2, 2]$ and a reward list $r\_list = [0, 2, 1, 3, 0]$ (Reprinted with permission from [7]).

6.3.2 Ant Colony Optimization (ACO) Engine

ACO is a probabilistic method to solve optimization problems that can be reduced to finding good paths in directed graphs, called ACO graphs. FlowTuner adopts ACO to solve the individual stage tuning. Given a design flow, multiple ACO engines are instantiated (see Figure 6.1), each of which optimizes a specific design stage, e.g., logic synthesis. The multiple ACO engines are

Algorithm 1: Cooperative co-evolutionary controller

Input: M: number of stages; L: number of epochs;
$n_1, n_2, \ldots, n_L$: number of trials in each epoch;
$\boldsymbol{w}_0 = [w_0, w_1, \ldots, w_M]$, $\sum_i w_i = 1$: initial stage importance;
$\mathbf{F}_0 = [\boldsymbol{f}_1^0, \boldsymbol{f}_2^0, \ldots, \boldsymbol{f}_M^0]$: initial configuration

Output: r_{best}: best reward obtained
$\mathbf{R} = [\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_M]$: best parameter configuration

Initialize: $r_{\text{best}} \leftarrow$ initial reward; $\mathbf{R} \leftarrow \mathbf{F}_0$; $\boldsymbol{w} \leftarrow \boldsymbol{w}_0$
`/* Perform` L `tuning epochs */`
for $l \leftarrow 1$ *to* L **do**
  `/* Allocate resource based on stage importance */`
  $\boldsymbol{a} \leftarrow [a_1, a_2, \ldots, a_M]$, where $a_i = \text{round}(w_i n_l)$ is the number of trials allocated to stage i, and round(x) means rounding x to an integer
  $r_{\text{temp}} \leftarrow r_{\text{best}}$
  stage list $s\_list = [\,]$; reward list $r\_list = [\,]$
  `/* Perform one epoch of stage-by-stage tuning, containing` $n_l = a_1 + a_2 + \ldots + a_M$ `trails */`
  for $i \leftarrow 1$ *to* M **do**
    `/* Perform` a_i `trails of tuning for stage` i `*/`
    for $j \leftarrow 1$ *to* a_i **do**
      `/* Suggest a new configuration */`
      $unvisited \leftarrow$ False
      while *not* $unvisited$ **do**
        The i-th ACO engine suggests a partial configuration $\boldsymbol{f}_i$
        $\mathbf{F} \leftarrow [\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{i-1}, \boldsymbol{f}_i, \boldsymbol{r}_{i+1} \ldots, \boldsymbol{r}_M]$
        if $\mathbf{F}$ *has not been visited* **then**
          $unvisited \leftarrow$ True
      Pass $\mathbf{F}$ to the Flow Controller to get the final QoR
      `/* Parse QoR and update ACO engine */`
      if *final QoR is obtained* **then**
        Calculate reward r according to final QoR
        Update the pheromones of the i-th ACO engine based on r
        if $r > r_{\text{best}}$ **then**
          $r_{\text{best}} \leftarrow r$; $\mathbf{R} \leftarrow \mathbf{F}$
        `/* Calculate reward improvement */`
        $r_{\text{imp}} \leftarrow \max(r - r_{\text{temp}}, 0)$
        Append i to s_list and r_{imp} to r_list
    $r_{\text{temp}} \leftarrow r_{\text{best}}$
  `/* Update the stage importance */`
  for $i \leftarrow 1$ *to* M **do**
    $u_i \leftarrow$ F-AUC of stage i, computed from s_list and r_list
    $w_i \leftarrow (w_i + u_i)/2$

orchestrated by the CCC as we described in the previous section.

We build the ACO graph in each of the ACO engines as follows.

- Node: Except the start node (node 0), each node corresponds to one group of highly-interacting parameters. Hence, the ACO graph for a design stage with n parameter groups contains $n+1$ nodes. The reason for mapping a group of highly coupled parameters, instead of one parameter, to a node is that we want to tune heavily-interacting parameters as a whole.

- Edge: For node i corresponding to the i-th parameter group with m feasible configurations, there exist m directed edges connecting from node $i-1$ to node i. Each edge corresponds to one parameter configuration for parameter group i.

- Path: Every path from the start node the the last node goes through all nodes exactly once, choosing an incoming edge for each node. Hence, each such path corresponds to a complete parameter configuration of the corresponding design stage.

Figure 6.6(a)–(b) shows an example of ACO graph creation. Node v_1' corresponds to parameter group (p_1, p_2); edges e_{11} through e_{14} represent the possible parameter configurations of the parameter group, e.g., e_{11} represents $(p_1, p_2) = (0, 0)$.

Each edge e_{ij}, the j-th incoming edge for node i, is associated with a pheromone level, denoted by τ_{ij}. The pheromone level in an edge indicates the fitness of the corresponding configuration for a parameter group. When to suggest a new configuration for the corresponding design stage (Line 9 in Algorithm 1), an *artificial ant* constructs a path from the start node to the last node in the ACO graph, producing a new parameter configuration to try. The ant chooses edges with probability proportional to the pheromone levels. In other words, the higher the pheromone level in an edge, the higher probability the edge will be selected.

When the reward is obtained for a path, the pheromone levels of all edges are updated in a positive-feedback manner (Line 19 in Algorithm 1):

$$\tau_{ij} = (1-\rho)\tau_{ij} + I(i,j)\varepsilon, \tag{6.1}$$

where $\rho \in [0, 1]$ is the evaporation ratio, $I(i, j)$ indicates whether the edge e_{ij} participates in the path (1 if e_{ij} belongs to the path; 0 otherwise), and ε is proportional to the reward of the path. We can see that the pheromone levels of edges participating a path with a positive reward will be raised up. In turn, these edges, corresponding to specific configurations of parameter groups, are more likely to be chosen in future trials since the pheromone levels are higher.

It is noteworthy that the cooperative co-evolutionary framework is flexible enough to work with a wide range of other searching engines. Even different searching engines can be deployed for different stages. We choose the ACO engine mainly for three reasons. First, ACO algorithms have been empirically proved to be effective for combinatorial optimization problems. Secondly, thanks to the evaporation ratio ρ in the pheromone update rule Equation (6.1), ACO algorithms are adaptive to dynamic environment by gradually forgetting outdated information, which is very important for our application. Because from the aspect of a design stage, its environment is formed by the EDA flow and parameter configurations in remaining stages, which is continuously changing. Lastly, pheromone levels in an ACO graphs can be initialized properly to boost the performance of ACO algorithms, which opens up opportunities for our knowledge transfer method introduced in Section 6.5.

6.4 Flow Jump Start and Early Stop

The flow controller launches a new flow thread for each parameter configuration $\mathbf{F}$ suggested by CC-ACO. Algorithm 2 outlines how the flow controller works. For efficient computation resource management, it exploits *flow jump start* that allows flow threads to start from intermediate results of previous trials along with *flow early stop* that terminates non-promising threads prematurely. We detail the flow jump start and early-stop capability of the flow controller in the following subsections.

6.4.1 Flow Jump Start

Parameter configurations and design outcomes of previous trials are stored in the result repository by the flow controller, as shown in Figure 6.1. For a new configuration $\mathbf{F}$, the controller first

Algorithm 2: Flow controller

Input: $\mathbf{F} = \{\boldsymbol{f}_1, \boldsymbol{f}_2 \ldots, \boldsymbol{f}_M, \}$: a new configuration;
r_{best}: best reward obtained;
$\Lambda = \{\mathbf{F}_1 : SynRe1; \ \mathbf{F}_2 : SynRe2 \ \ldots\}$: repository of design outcomes, where $\mathbf{F}_i, i = 1, 2 \ldots$ is the configuration of a previous trial and $SynRe$ is its outcome;
$\Phi = \{\phi_1, \phi_2 \ldots, \phi_{M-1}\}$: a set of models for reward upper bound prediction, where $\phi_i, i = 1, 2 \ldots, M-1$, is the model for predicting the final (post-route) reward upper bound after stage i;
$\Omega = \{\omega_1, \omega_2 \ldots, \omega_{M-1}\}$: collection of QoR data, where ω_i contains a set of data pairs (QoR after stage i, final QoR) for the training/parameter estimation of prediction model ϕ_i

Output: final QoR or None;
Updated r_{best}, Λ, Φ and Ω

`/* Flow jump start */`

Query the collection Λ to find $k^* = \max_{k \in [2,M]} k$
s.t., $\exists \mathbf{F}' = \{\boldsymbol{f}'_1, \boldsymbol{f}'_2 \ldots, \boldsymbol{f}'_M\} \in \Lambda$ and $\forall i \in [1, k-1], \boldsymbol{f}'_i = \boldsymbol{f}_i$
if k^* *can be found successfully* **then**
    Fetch the synthesis result after the $(k^* - 1)$-th stage for $\mathbf{F}'$ from Λ
else
    $k^* \leftarrow 1$
Start a new flow thread from the k^*-th stage

`/* Flow early stop */`

for $i \leftarrow k^*$ *to* $M - 1$ **do**
    Run the i-th stage and collect intermediate QoR after stage i
    `/* FlowTuner does not conduct early stop until there exist enough data pairs in` ω_i `to train` ϕ_i `to estimate the reward bound effectively */`
    if $|\omega_i| < 10$ **then**
        continue
    Feed the intermediate QoR to model ϕ_i to get an estimation of reward upper bound $\hat{r}$
    if $\hat{r} < r_{best}$ **then**
        return None, r_{best}, Λ, Φ and Ω

`/* Update` r_best, Λ, Φ `and` Ω `*/`

Run the M-th stage and collect the final QoR r
if $r > r_{best}$ **then**
    $r_{best} \leftarrow r$
Add synthesis results and QoR results to Λ and Ω, respectively
Update models in Φ with dataset Ω
return final QoR, r_{best}, Λ, Φ and Ω

queries if there exists a previous configuration $\mathbf{F}'$ and a constant $k \in (1, M]$ that satisfies $\boldsymbol{f}_i = \boldsymbol{f}'_i$ for all $i \in [1, k-1]$. If so, find the maximum k, denoted as k^*. Then the intermediate result of the $k^* - 1$-th stage for $\mathbf{F}'$ is fetched and a new flow thread starts from the k^*-th stage. The *jump-start rate* is defined as the ratio of the number of times where an archived configuration is found to the number of total trials. The stage-by-stage tuning behavior of the CC-ACO engine in each epoch produces higher jump-start rates as stages proceed with more trials.

6.4.2 Flow Early Stop

6.4.2.1 Branch-and-Bound Strategy

Whenever a design stage completes, flow threads send intermediate results to the flow controller and query whether to move on to the next stage. A partial parameter configuration $[\boldsymbol{f}_1, \ldots, \boldsymbol{f}_k]$ ($k \in [1, M]$) is analogous to a branch in a decision tree. At each stage, the flow controller estimates a probabilistic upper bound of *final* reward of the branch and compares it with the current best reward. If the upper bound is inferior, the thread is terminated. Otherwise, it proceeds to the next stage. No computation resources are assigned to the terminated branches in the following trials.

6.4.2.2 Reward Upper Bound Estimation

The reward upper bound estimation is made by summing up the upper bounds of individual QoR metrics of interest. Ignoring the dependencies among metrics tends to increase the "safety" of the estimated upper bounds, because it is hard to improve all of the QoR metrics simultaneously.

Figure 6.4 demonstrates the process of estimating the upper bound for a QoR metric. An estimate of the final QoR is obtained by a linear regression model using the intermediate QoR as input, which captures the correlation between different stages. The upper bound estimation is obtained by adding a guard band to the estimated final QoR. As shown in Figure 6.5, the error margins of the linear regression model diminish as stages progress. For example, the prediction of final QoR at the clock-tree-synthesis (CTS) stage is more accurate than the prediction at the placement stage. It suggests that the guard band should be adaptive to the prediction errors at

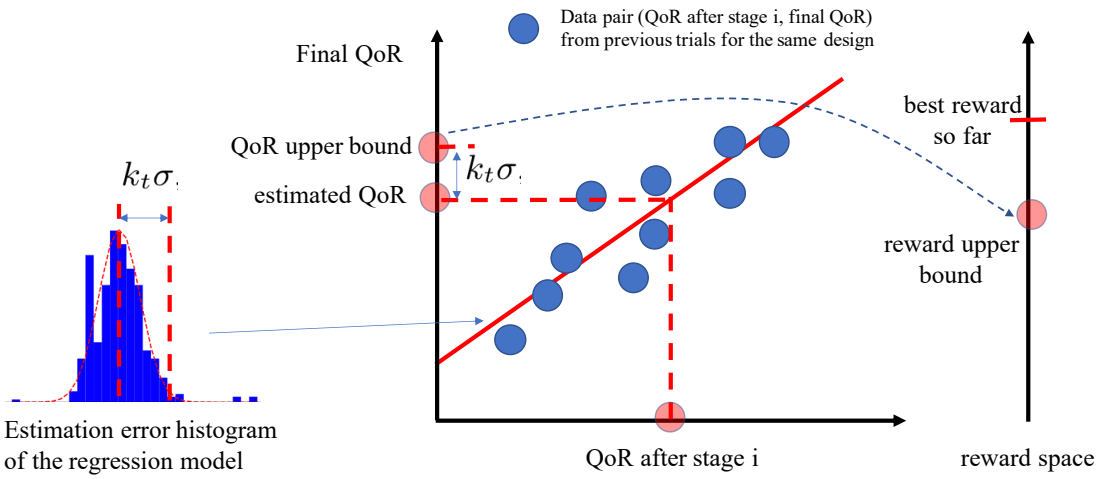


Figure 6.4: Reward upper bound estimation. Final QoR is estimated by a linear regression model utilizing the correlation between different stages. A guard band is determined by analyzing the estimation errors of the regression model. The final QoR upper bound is obtained by adding the guard band to the QoR prediction result, which is then mapped to the reward upper bound according to the reward function (Reprinted with permission from [7]).

different stages. Depicted in Figure 6.5, the prediction errors of linear regression models can be described by a zero-mean Gaussian distribution. With the Gaussian model of prediction errors, the guard band is calculated based on properties of Gaussian distribution [123]. To be more specific, the guard band is set to be $k_t\sigma$, where σ is the standard deviation of the prediction errors and k_t is a hyper-parameter controlling the tradeoff between QoR results and parameter tuning runtime. The smaller the k_t is, the more likely a branch is pruned to reduce runtime at risk of missing good configurations.

For each QoR metric and each design stage, our upper bound prediction approach has three learnable parameters: (1) the slope and (2) the off-set of the linear regression model, and (3) the standard deviation of the zero-mean Gaussian prediction errors. Data from previous trials for the same design is used to learn these parameters.

It is not easy to accurately predict the post-route QoR at early stages due to the complexity of EDA flows. Instead, we aim to provide a safe yet effective reward upper bound estimation by monitoring the prediction errors and adding an adaptive guard band to the prediction results. The simple linear regression models might not deliver tight upper bounds. However, such loose upper bounds already work very well with our branch-and-bound strategy, since the bounds are compared with

the best reward so far, which is often obviously higher than upper bounds of many configurations. Of course, the linear regression model can be easily replaced by other more advanced models that may provide tighter bounds, hence further boosting the efficiency of our early-stop approach.

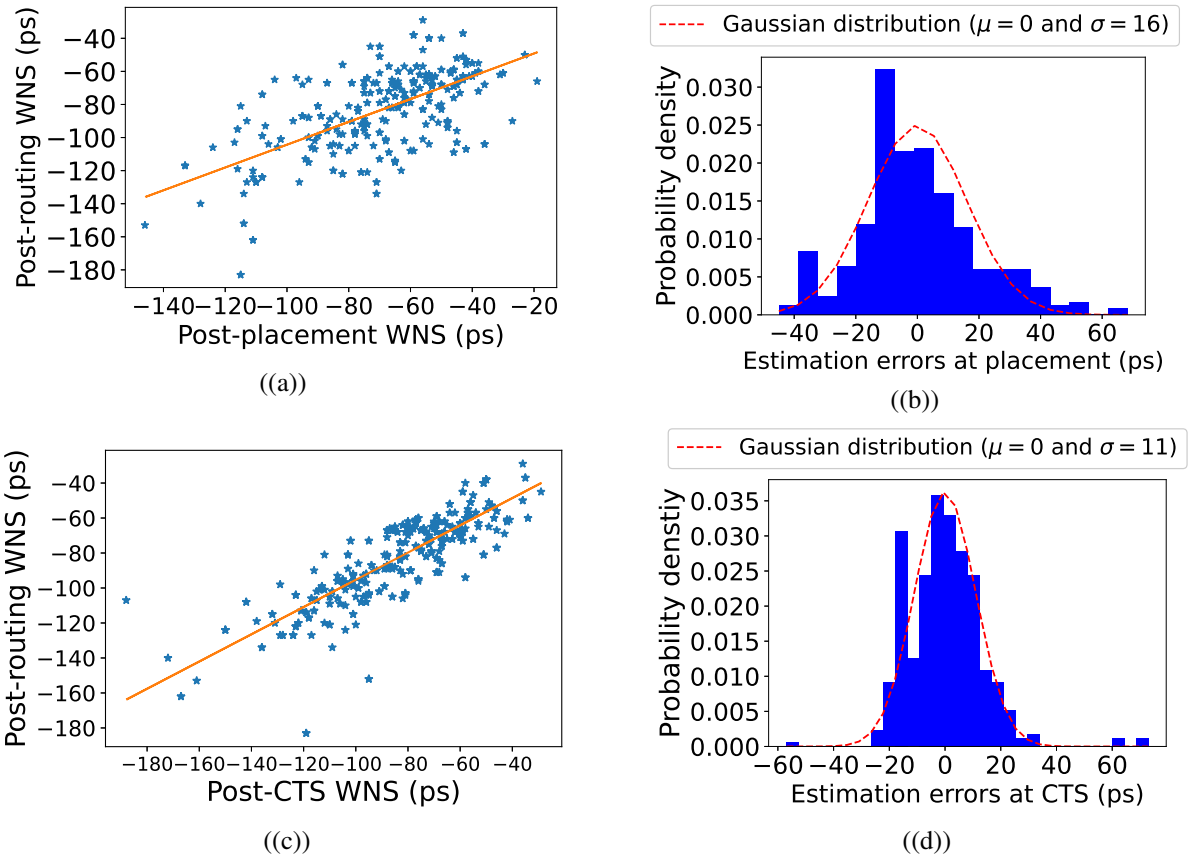


Figure 6.5: WNS (worst negative slack) correlation between different design stages: (a) post-placement and post-route, (b) estimation error histogram of a linear model at placement stage, (c) post-CTS and post-route, and (d) estimation error histogram at CTS. The design is wb_dma of IWLS 2005 benchmark, with 315 different parameter configurations (Reprinted with permission from [7]).

6.5 Knowledge Transfer and CC-ACO Warm-up

The stage importance update of CCC (Section 6.3.1) and pheromone level update for ACO graphs (Section 6.3.2) are essentially the learning of knowledge about the impact of flow parameters on designs for better resource utilization and efficient searching in the parameter configuration

space. FlowTuner accelerates the learning by extracting knowledge from previous design data and transferring it to new designs. Also, the transferred knowledge can guide the selection of parameters to be tuned and construction of ACO graphs, to narrow down the search space and enhance pheromone update efficiency. In the following, we define three types of parameter knowledge and show how to exploit them to warm-up the CC-ACO engine.

6.5.1 Legacy Data Preparation

The EDA flow can be divided into multiple stages upon designers' needs. For instance, in our experiment the flow is divided into 7 stages: logic synthesis, placement, post-placement optimization, CTS, post-CTS optimization, routing and post-route optimization. All parameter combinations in each stage are enumerated, and the final QoR for each combination is measured while using the baseline settings[§] for the remaining stages. In such a way we generate around three hundreds samples for each of 11 legacy designs. In general, the similarity between legacy designs and new designs in terms of how flow parameters affect QoR determines the effectiveness of our knowledge transfer method. However, it is difficult to define the similarity. Hence, we try to extract *general* knowledge that is applicable to a wide range of designs by aggregating the parameter knowledge extracted from a set of legacy designs. In our experiment, more than 3500 design samples from 11 legacy designs were collected. The knowledge extracted from them has been empirically proven to be beneficial for the tuning of 5 new designs.

6.5.2 Parameter Importance

Different parameters have different degrees of impact on the final QoR of a design. For effective flow parameter tuning, parameters having large positive impact on QoR need to be explored more extensively. To quantitatively measure such importance of individual parameters, we introduce a measure called *parameter importance*, which we define as follows. Given a parameter f whose

[§]Users can define baseline settings as needed, such as using the highest effort settings suggested by EDA tool manuals.

baseline setting is $\bar{f}$, the reward improvement r_{imp} is evaluated by

$$r_{\text{imp}}(f) = \max\{0,\ r(\ldots, f, \ldots) - \bar{r}\}, \tag{6.2}$$

where $r(*)$ is the reward function, and $\bar{r}$ is the baseline reward obtained by using baseline settings for all parameters. The parameter importance of f, denoted by $\text{IM}(f)$, is then given by

$$\text{IM}(f) = \max_{\Delta f}\ r_{\text{imp}}(\ldots, \bar{f} + \Delta f, \ldots) - r_{\text{imp}}(\ldots, \bar{f}, \ldots), \tag{6.3}$$

where Δf is a perturbation on parameter f.

When measuring the importance for one parameter, we vary the settings for the remaining parameters and obtain the maximal IM value. Next, we average the results of legacy designs. Our definition of parameter importance has the following characteristics:

- Unlike variance-based importance metrics [58] that consider both positive and negative effects on rewards obtained by parameter perturbation, our measure focuses on reward improvement over the baseline reward. Our measure naturally better fits to the parameter tuning scenario, where the reward improvement is a primary concern.
- Our measure focuses on perturbations from the baseline setting. As baseline settings can lead to decent QoRs and we care about the significance and necessity of further fine-tuning.

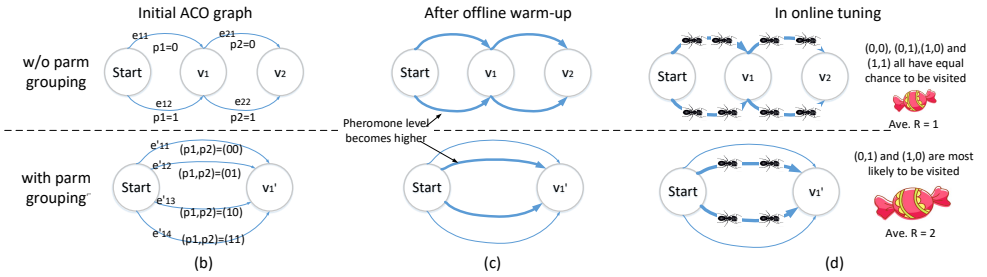
The stage importance defined in Section 6.3.1 can be viewed as an extension of parameter importance. And the stage importance can also be calculated with legacy design data using the method shown in Section 6.3.1.

6.5.3 Parameter Bias

The likelihood for a parameter configuration to produce higher QoRs can be inferred by analyzing the archives of legacy designs. FlowTuner actively biases the ACO engine search process by initializing pheromone levels of ACO graphs using (*parameter configuration* and *final QoR*) pairs from the legacy design archives.

Parms	Reward	
p1, p2	Legacy design	New design
0, 0	0	0
0, 1	1	1.5
1, 0	1	2.5
1, 1	0	0

(a)



(b) (c) (d)

Figure 6.6: Parameter grouping and pheromone level initialization in ACO graphs. (a) An example tuning problem with two parameters and the rewards of different parameter settings. (b) ACO graphs with and without parameter grouping. (c) Offline warm-up for updating the pheromone levels of each edge according to the rewards of the legacy design. (d) Online tuning based on the pheromone levels, where edges with higher pheromone levels are more likely to be visited (Reprinted with permission from [7]).

Each legacy data sample corresponds to a parameter configuration that is different from the baseline configuration for at least one stage, assumed to be stage k. Denote its configuration for stage k as $\boldsymbol{f}_k$, which corresponds to a path in the ACO graph for stage k, as illustrated in Section 6.3.2. Then we can update the pheromone levels of the ACO graph according to the pheromone update rule Equation (6.1), where ϵ is set to be proportional to the reward calculated based on the final QoR and ρ is set to be 0 here.

6.5.4 Parameter Interaction

Some parameters have strong interactions with each other in affecting the final QoR. FlowTuner captures such parameter interactions by detecting the violation of the monotonicity condition [126]. Given a parameter configuration $\boldsymbol{f}_k$ for stage k and its two parameters f_i^k and f_j^k, we first compute three measures:

$$\Delta_i^k = r(\ldots, f_i^k + \Delta f_i^k, \ldots) - r(\ldots, f_i^k, \ldots),$$
$$\Delta_j^k = r(\ldots, f_j^k + \Delta f_j^k, \ldots) - r(\ldots, f_j^k, \ldots),$$
$$\Delta_{i,j}^k = r(\ldots, f_i^k + \Delta f_i^k, \ldots, f_j^k + \Delta f_j^k, \ldots) - r(\ldots, f_i^k, \ldots, f_j^k, \ldots),$$

where Δ_i^k, Δ_j^k and $\Delta_{i,j}^k$ are the changes in reward function $r(*)$ caused by perturbation of the f_i^k, f_j^k, and both, respectively. For $\Delta_i^k > 0$ and $\Delta_j^k > 0$, the monotonicity of the two parameters holds

if and only if $\Delta^k_{i,j} > \Delta^k_i$ and $\Delta^k_{i,j} > \Delta^k_j$.

If the monotonicity condition does not hold, the degree of interaction between two parameters, $\theta^k_{i,j}$ is estimated by

$$\theta^k_{i,j} = \max\{\Delta^k_i - \Delta^k_{i,j}, \Delta^k_j - \Delta^k_{i,j}, 0\}. \tag{6.4}$$

Using the legacy design archives, FlowTuner calculates the degree of interaction for every pair of parameters within a stage. Top-ranking pairs is then consolidated into parameter groups, and the knowledge is utilized when building the ACO graphs.

6.5.5 CC-ACO Warm-Up with Transferred Parameter Knowledge

FlowTuner exploits the extracted knowledge to warm-up the CC-ACO engine offline in the following ways:

1. Selection of important parameters. Given a set of candidate tuning parameters suggested by designers, we calculate the importance of every parameter with legacy design data according to Equation (6.3) and 6.2. Only top-ranking ones are deployed for online tuning, which greatly narrow down the search space.

2. Parameter grouping. For parameters in one design stage, their pair-wise parameter interaction is computed according to Equation (6.4). Parameters with high parameter interaction are consolidated into groups, and each parameter group corresponds to one node in the ACO graph. Figure 6.6 gives an example of how parameter grouping helps enhance pheromone update efficiency in ACO graphs.

3. Initialization of stage importance. F-AUC-based stage importance can be calculated with legacy design data using the method shown in Section 6.3.1. We apply the extracted knowledge of stage importance to initialize the stage importance vector $\boldsymbol{w}_0$ in Algorithm 1 for better computation resource utilization.

4. Initialization of pheromone levels in ACO graphs. We initialize pheromone levels of ACO graphs using (*parameter configuration* and *final QoR*) pairs from the legacy design archives.

The warmed-up pheromone levels will actively biases the ACO engine online search process.

Figure 6.6(a)–(d) illustrates how 2) parameter grouping and 4) initialization of pheromone levels work. Consider two parameters that strongly interact with each other and a new design having a similar parameter bias to the legacy design (see Figure 6.6(a)). The highly interactive parameters are combined into a group, which corresponds to a single node in the ACO graph (Figure 6.6(b)). During the ACO warm-up, the pheromone levels for edges are updated using the rewards of different parameter settings on the legacy design (Figure 6.6(c)). Without the parameter grouping, the low-reward setting $(0, 0)$ and $(1, 1)$ have positive pheromone levels. Thus, they are likely to be visited during the subsequent parameter search. By contrast, with the grouping, they have 0 pheromone levels (Figure 6.6(d)). Hence, the high-reward settings ($(0, 1)$ and $(1, 0)$ in this case) are more likely to be tried than the low-reward settings. As such, it makes the search process more efficient, leading to a better solution within given runtime budget.

6.6 Single-Trade-Off Tuning Experimental Evaluation

6.6.1 Experimental Setup

Our experiments were conducted on 16 IWLS 2005 benchmark designs [114], whose characteristics are shown in Table 6.1. A commercial EDA flow with Cadence Genus for logic synthesis and Innovus for physical design was applied to these designs, targeting a 7nm technology node [115]. A total of 21 discrete parameters were tuned with a search space as large as 2448880128. Information about the parameters are shown in Table 6.2. Experiments were performed on linux servers equipped with Intel Xeon E5-2697A CPUs.

To evaluate the efficiency and the effectiveness of the proposed FlowTuner method, we also implemented the following flow tuning methods and compared the performance of them:

- **Baseline** setting is a highest-effort setting suggested by the EDA tool manuals. Details can be found in Table 6.2.
- **BO**: A Bayesian optimization approach similar to [119], implemented with the package [127].

Table 6.1: Benchmark Circuits (Reprinted with permission from [7]).

	ID	Name	#Cells	Clock Period (ps)
Legacy design	1	wb_dma	2792	280
	2	des3_area	3097	730
	3	systemcdes	3249	300
	4	systemcaes	5353	480
	5	mem_ctrl	6985	300
	6	ac97_ctrl	9467	180
	7	aes_core	14737	420
	8	b17	15696	470
	9	wb_conmax	18753	600
	10	b18	27113	810
	11	fpu_double	49123	330
Testing design	12	ethernet	10990	280
	13	b19	48398	1200
	14	vga_lcd	59498	320
	15	FFT_64	59551	500
	16	FFT_128	125558	360

- **Recommender**: A recommender system based on [57]. A latent factor model was trained offline. The sampling in the first 30 online iterations was based on the top-30 high reward configurations from legacy design archives, whose QoRs were used to fine-train the model. The sampling in the last 20 iterations was based on the recommendation of the trained model.

- **FIST**: A parameter-importance and XGboost-based sampling approach similar to [58].

To demonstrate the impacts of our proposed knowledge transfer method and the cooperative co-evolutionary framework, we also implemented the following methods:

- **FlowTuner w/o know.**: FlowTuner without knowledge transfer.

- **ACO w/ know.**: An ACO engine enhanced by knowledge transfer for tuning whole EDA flow. This is to demonstrate the effects of the cooperative co-evolutionary framework.

- **ACO w/o know.**: An ACO engine for the whole flow, same as the above, but without knowledge transfer.

We prepared legacy data as shown in Section 6.5 for the Recommender, FIST, and FlowTuner methods. The Recommender method requires a large volume of legacy data. Thus, the data col-

lected from the tuning with all other methods were used as additional legacy data for it. We used the following reward function:

$$r = w_0 \frac{\mathrm{TNS}_{\mathrm{ref}} - \mathrm{TNS}}{\mathrm{TNS}_{\mathrm{ref}}} + w_1 \frac{\mathrm{WNS}_{\mathrm{ref}} - \mathrm{WNS}}{\mathrm{WNS}_{\mathrm{ref}}} + w_2 \frac{\mathrm{P}_{\mathrm{ref}} - \mathrm{P}}{\mathrm{P}_{\mathrm{ref}}}, \tag{6.5}$$

where TNS, WNS and P are the total negative slack, worst negative slack and the total power of a design instance, respectively; $\mathrm{TNS}_{\mathrm{ref}}$, $\mathrm{WNS}_{\mathrm{ref}}$ and $\mathrm{P}_{\mathrm{ref}}$ are the reference values obtained with baseline settings; and the weights w_0, w_1 and w_2 control the trade-off between these QoR metrics. We used $1, 1$ and 10 for w_0, w_1 and w_2, respectively, since we empirically found that TNS and WNS reductions achieved by parameter tuning are about one order larger than the power reduction for our benchmark designs. Note that other QoR metrics can be easily incorporated by changing (6.6). A total of **50** trials were conducted for each method. And the runtime for 50 trials were measured.

6.6.2 Results

6.6.2.1 Parameter Knowledge Extraction Results

Parameter importance and stage importance results are shown in Figure 6.7. We can find that parameters in early stages tend to have a higher impact on final QoR than those in late stages; while a few parameters in late stages are also influential, such as the *powerEffort* in the post-route optimization stage. The most significant parameter *syn_opt_effort* has importance 2.037. We used $0.1 \times 2.037 = 0.204$ as the threshold to choose the top-ranking parameters, as shown in Figure 6.7, for the online tuning in our FlowTuner system. It greatly reduced the search space. According to our definition of importance in (Equation (6.3)), fine-tuning a parameter with low importance from its baseline setting is less likely to bring QoR improvement. Hence, we used baseline settings for non-selected parameters.

For the top-ranking parameter selected in the previous step, we performed parameter grouping according to the parameter interaction results. For the 5 parameters in the logic synthesis stage, p_1, p_2 and p_3 were consolidated into one parameter group, while p_4 and p_5 became the second and the third group, respectively, since (p_1, p_2) and (p_1, p_3) have the strongest pair-wise interactions in

Table 6.2: Flow Parameters with Baseline settings highlighted in Bold (Reprinted with permission from [7]).

Stage	ID	Parameter	Option
syn	1	syn_generic_effort	**high**, medium, low
	2	syn_map_effort	**high**, medium, low
	3	tns_critical_range	0, 8, **15**
	4	optimize_net_area	**true**, false
	5	syn_opt_effort	**extreme**, high, medium, low
place	6	place_detail_wire_ length_opt_effort	**high**, medium, none
	7	place_global_cong_effort	auto, **high**, medium, low
	8	place_global_max_density	1.0, 0.9, **0.8**
place_opt	9	maxDensity	**0.95**, 0.9, 0.8
	10	powerEffort	none, low, **high**
	11	reclaimArea	**true**, false
cts	12	auto_limit_insertion_delay	**1.4**, 1.2, 1.0
	13	cell_density	0.95, 0.9, **0.8**
cts_opt	14	maxDensity	**0.95**, 0.9, 0.8
	15	powerEffort	none, low, **high**
	16	reclaimArea	**true**, false
route	17	postRouteReclaim	none, setuoAware, **holdAndSetupAware**
	18	routeWithTimingDriven	**true**, false
route_opt	19	maxDensity	**0.95**, 0.9, 0.8
	20	powerEffort	none, low, **high**
	21	reclaimArea	**true**, false

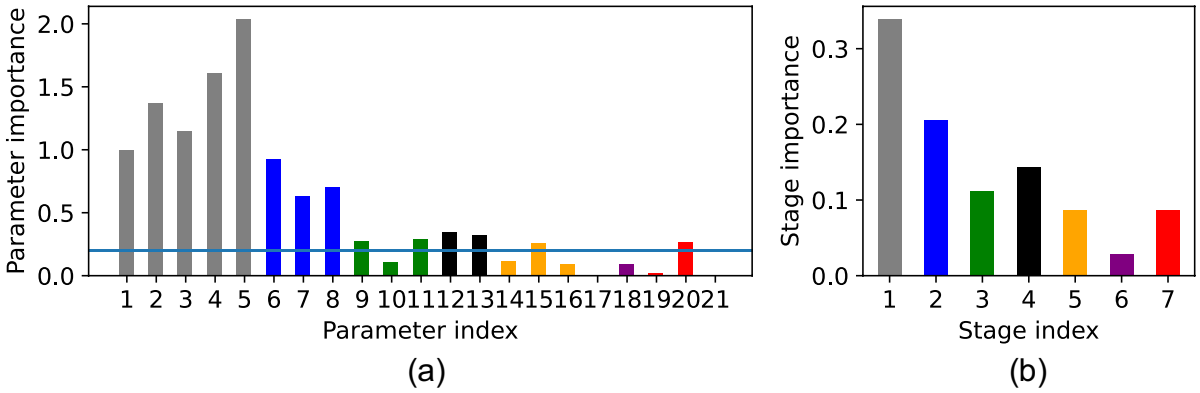


Figure 6.7: Parameter importance and stage importance extracted from legacy designs: (a) parameter importance, (b) stage importance (Reprinted with permission from [7]).

Table 6.3: QoR Comparison with Different Parameter Tuning Methods (Reprinted with permission from [7]).

Des.	Metric	Baseline	BO [119]	Recom. [57]	FIST [58]	ACO w/o know.	ACO w/ know.	FlowTuner w/o know.	**FlowTuner**
12	TNS/ns	-5.028	-3.502 (30.35%)	-3.398 (32.42%)	-2.035 (59.53%)	-2.975 (40.83%)	-1.590 (68.38%)	-2.425 (51.77%)	**-1.299** (74.16%)
	WNS/ps	-50	-46 (8.00%)	-28 (44.00%)	-31 (38.00%)	-36 (28.00%)	-26 (48%)	-30 (40.00%)	**-22** (56.00%)
	P/mW	19.346	19.082 (1.36%)	19.101 (1.27%)	19.199 (0.76%)	19.129 (1.12%)	19.219 (0.66%)	19.153 (1.00%)	**18.878** (2.42%)
13	TNS/ns	-0.463	**-0.005** (98.92%)	-0.010 (97.84%)	-0.011 (97.62%)	-0.189 (59.18%)	-0.010 (97.84%)	-0.061 (86.83%)	-0.009 (98.06%)
	WNS/ps	-32	**-1** (96.88%)	-2 (93.75%)	-2 (93.75%)	-15 (53.13%)	-2 (93.75%)	-6 (81.25%)	-2 (93.75%)
	P/mW	15.302	15.491 (-1.24%)	15.054 (1.62%)	15.430 (-0.84%)	15.223 (0.52%)	15.054 (1.62%)	15.285 (0.11%)	**15.028** (1.79%)
14	TNS/ns	-8.896	-0.976 (89.03%)	-0.577 (93.51%)	-0.952 (89.30%)	-6.499 (26.94%)	-1.072 (87.95%)	-0.591 (93.36%)	**-0.540** (93.93%)
	WNS/ps	-27	-25 (7.41%)	-29 (-7.41%)	-12 (55.56%)	-27 (0.00%)	-12 (55.56%)	-33 (-22.22%)	**-10** (62.96%)
	P/mW	116.035	103.668 (10.81%)	103.008 (11.23%)	103.292 (10.98%)	107.768 (7.12%)	**102.744** (11.45%)	105.857 (8.77%)	102.850 (11.36%)
15	TNS/ns	-0.597	-0.005 (99.16%)	-0.047 (92.13%)	-0.014 (97.65%)	-0.004 (99.33%)	-0.007 (98.83%)	-0.020 (96.65%)	**-0.003** (99.50%)
	WNS/ps	-22	**-1** (95.45%)	-9 (11.11%)	-7 (22.22%)	-2 (90.91%)	-3 (86.36%)	-4 (81.82%)	-3 (86.36%)
	P/mW	67.358	66.073 (1.91%)	67.57 (-0.31%)	65.188 (3.22%)	71.475 (-6.11%)	66.218 (1.69%)	**64.833** (3.75%)	65.959 (2.08%)
16	TNS/ns	-2.092	-0.344 (83.56%)	-0.448 (78.59%)	-0.610 (70.84%)	-0.324 (84.51%)	-0.242 (88.43%)	0.148 (92.93%)	**-0.101** (95.17%)
	WNS/ps	-18	-12 (33.33%)	-16 (11.11%)	-14 (22.22%)	-10 (44.44%)	-12 (33.33%)	-10 (44.44%)	**-9** (50.00%)
	P/mW	193.228	190.773 (1.27%)	188.739 (2.32%)	191.803 (0.74%)	193.206 (0.00%)	**188.606** (2.39%)	192.647 (0.30%)	190.324 (1.50%)
Average TNS reduction			80.20%	78.90%	82.99%	62.16%	88.29%	84.31%	**92.16%**
Average WNS reduction			48.21%	40.19%	55.54%	43.30%	63.40%	45.06%	**69.82%**
Average Power reduction			2.82%	3.23%	2.97%	0.53%	3.56%	2.79%	**3.83%**

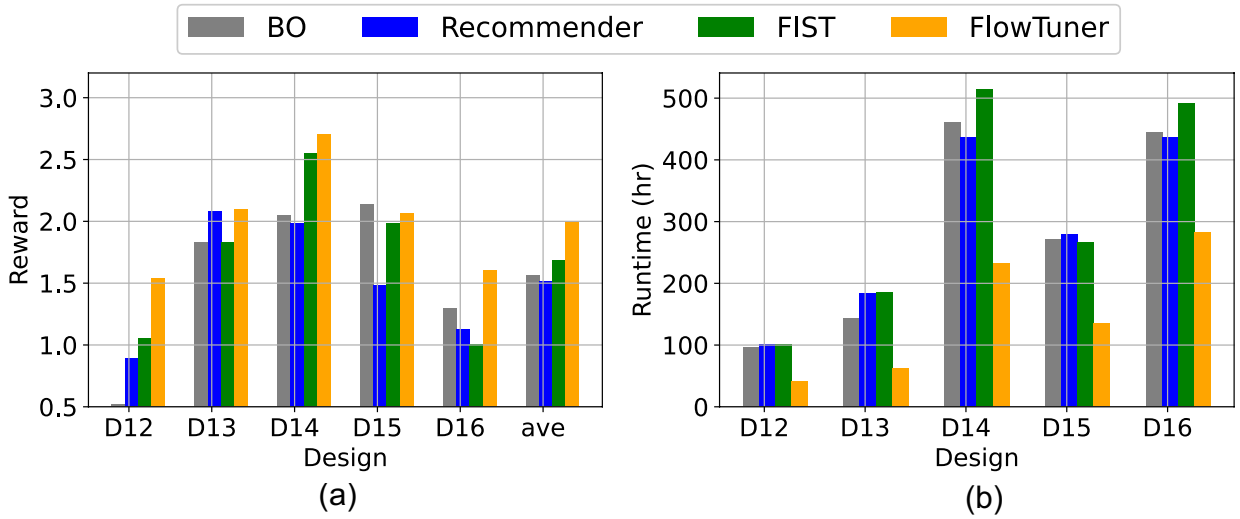


Figure 6.8: Comparison with previous tuners: (a) reward and (b) runtime (Reprinted with permission from [7]).

this stage based on analysis results of Equation (6.4). Similarly, p7 and p8 were consolidated into a group, and each of the remaining selected parameters formed one group. Based on parameter grouping results, an ACO graph was constructed for each design stage, and pheromone levels of

the ACO graphs were initialized as described in Section 6.5.3.

To verify the transferability of parameter knowledge extracted from legacy designs, the following extension experiment was conducted. We generated 315 instances with different configurations for the largest testing design, FFT_128, from whose QoR results parameter knowledge was extracted. Then the similarity between knowledge from FFT_128 and that from legacy designs was analyzed. The correlation coefficient (CoR) between parameter importance scores from FFT_128 and those from legacy designs, the CoR between stage importance scores and the CoR between parameter interaction scores were 0.79, 0.80 and 0.84, respectively. The strong correlations means that, though the testing designs are larger than legacy designs, parameter knowledge extracted from legacy designs are still transferable and useful. However, due to the diversity of designs and complexity of EDA flows, the correlations were far from being perfect. It implies that, besides utilizing exploration via knowledge transfer, intelligent tuners should also perform efficient exploration on new designs.

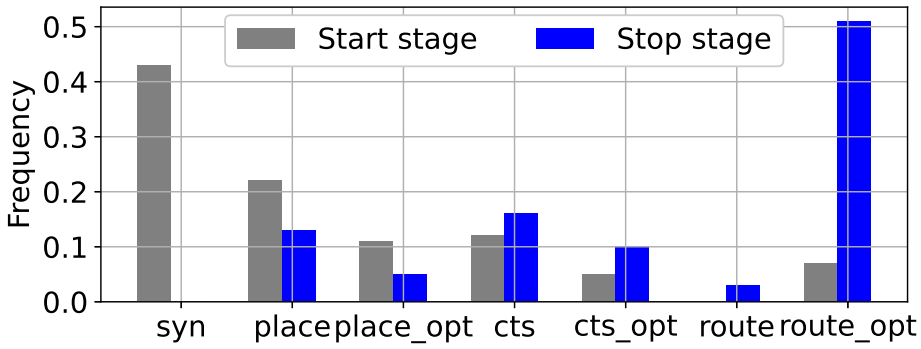


Figure 6.9: Jump-start and early-stop frequencies (Reprinted with permission from [7]).

6.6.2.2 Online Tuning Results

Table 6.3 shows the TNS, WNS and power reductions obtained after the parameter tuning. Figure 6.8 compares the final reward values and runtime of 50 tuning trials of FlowTuner with previous methods. It can be observed that FlowTuner achieved the highest rewards in 4 of the 5

testing designs, obtaining around 19% higher average reward against previous methods, in 50% less runtime. In terms of QoR improvement, FlowTuner delivered 11% higher average TNS reduction, 25.7% higher average WNS reduction and 19% higher average power reduction than previous methods.

The BO method does not leverage knowledge from legacy designs, and the transferred knowledge is limited to parameter importance in the FIST method, while the Recommender method under-emphasizes exploration on new designs. The superior performance of FowTuner comes from both our knowledge transfer techniques and effective exploration with the CC framework. Without knowledge transfer (i.e., Flowtuner w/o know.), without the CC framework (ACO w/ know), and without both (ACO w/o know.), the average reward of FlowTuner across 5 testing designs dropped by 22%, 7% and 45%, respectively. Table 6.3 demonstrates that the TNS, WNS and power improvements obtained after the parameter tuning decreased considerably without knowledge transfer and/or the CC framework.

Both flow jump-start and early-stop techniques contribute to the runtime efficiency of FlowTuner. Figure 6.9 shows the statistical information of start and stop stages of tuning trials. It can be observed that 57% trials benefited from our jump-start technique and 49% trials were terminated prematurely to save runtime. Note that no tuning trials started with the routing stage, since no routing parameters were selected for the online tuning in our FlowTuner method. No trials stopped at the logic synthesis stage due to the weak correlation between the QoR after logic synthesis and post-route QoR.

Best parameter configurations found for 5 testing designs by FlowTuner are listed in Table 6.4. Only the top-ranking parameters from the importance analysis are shown in the table. We can see the diversity of best configurations for different designs, which implies the necessity of effective exploration for new designs. Some patterns can also be observed, such as some parameters (e.g., syn_generic_effort) needs to be explored more extensively than others (e.g., place_global_max_density) and some parameters have obvious bias in their configurations (e.g., setting syn_opt_effort to extreme or high is more likely to deliver high QoR than setting it to low),

Table 6.4: Best Parameter Configurations Obtained by FlowTuner.

ID	Parameter	D12	D13	D14	D15	D16
1	syn_generic_effort	low	med.	high	high	med.
2	syn_map_effort	high	med.	med.	low	med.
3	tns_critical_range	0	8	0	8	8
4	optimize_net_area	false	false	false	false	false
5	syn_opt_effort	high	ext.	high	med.	ext.
6	place_detail_wire _length_opt_effort	high	med.	med.	high	high
7	place_global_cong_effort	auto	auto	med.	auto	auto
8	place_global_max_density	1	1	1	1	1
9	maxDensity	0.95	0.95	0.95	0.95	0.8
11	reclaimArea	true	true	true	true	false
12	auto_limit_insertion_delay	1.4	1.4	1.4	1.4	1.4
13	cell_density	0.8	0.8	0.8	0.8	0.8
15	powerEffort	none	high	high	high	high
20	powerEffort	none	high	none	high	high

which confirms the effectiveness of our knowledge transfer techniques.

6.7 Multi-Trade-off Tuning and Evaluation

6.7.1 Pareto Front Exploration via Concurrent Multi-Trade-off Learning

In this section, we extend the CC-ACO engine to deal with competing QoR metrics effectively via multi-trade-off learning. To this end, we adopt the concept of the Pareto front exploration. The Pareto front consists of *non-dominated* points in the metric space. A non-dominated point indicates that there is no other point that is better in terms of at least one metric while no worse in the others. In other words, the Pareto front represents the best-possible QoRs. Decomposition-based methods [128] divide a Pareto front exploration problem into a set of individual sub-problems, which are solved simultaneously while allowing information exchange among them. We apply this concept to FlowTuner in order to enable efficient Pareto front exploration via concurrent multi-trade-off learning.

Here we take the optimization of the WNS and the total cell area as an example. The Pareto front exploration problem is decomposed into a set of single-trade-off problems, each of which

maximizes a weighted sum reward:

$$r(\mathrm{WNS}, A) = w_0(\mathrm{WNS} - \mathrm{WNS}_{\mathrm{ref}}) + w_1 \frac{A_{\mathrm{ref}} - A}{A_{\mathrm{ref}}}, \tag{6.6}$$

where WNS and A are the worst negative slack and the total cell area of a design instance, respectively; $\mathrm{WNS}_{\mathrm{ref}}$ and A_{ref} are the reference WNS and the reference area (e.g., obtained with default parameter setting), respectively; and the weights w_0 and w_1 control the trade-off between WNS and area. Each of the single-trade-off sub-problems sets different weights w_0 and w_1. Note that other QoR metrics, e.g., total power, can be easily incorporated by changing Equation (6.6).

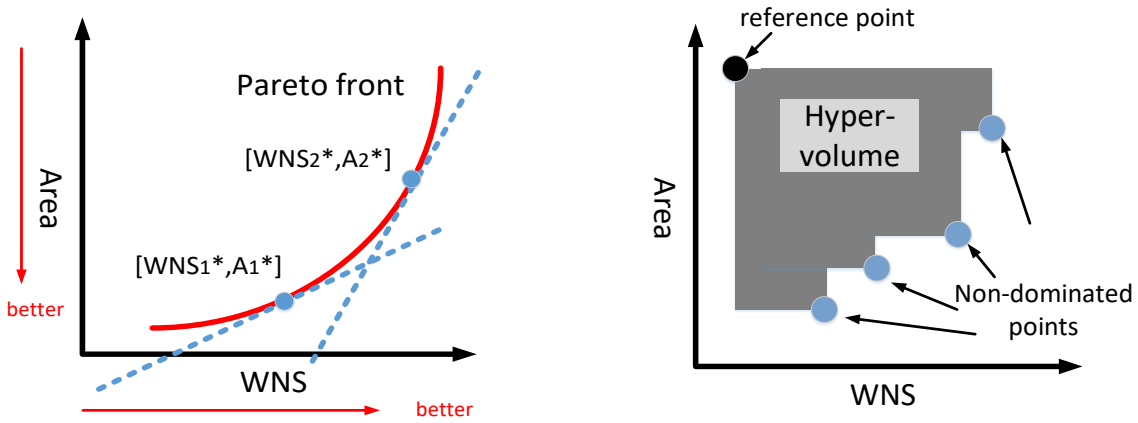
It has been proved in [129] that, as shown in Figure 6.10(a), the solution (WNS^*, A^*) for a single trade-off problem is actually the tangential point between the Pareto front and the straight line derived from Equation (6.6):

$$A = \frac{w_0 A_{\mathrm{ref}}}{w_1}\mathrm{WNS} + \frac{w_1 - w_0\mathrm{WNS}_{\mathrm{ref}} - r^*}{w_1} A_{\mathrm{ref}}, \tag{6.7}$$

where r^* is an optimal reward. The solutions for a set of single trade-off problems deliver a set of points on the Pareto front. In the following, we introduce the major changes to the CC-ACO engine to enable multiple trade-offs to be tackled simultaneously and allow concurrent learning for all trade-offs.

1) Every ACO engine is extended to maintain a set of ACO graphs $G_1, G_2, \ldots G_N$ instead of one. Each ACO graph corresponds to one of the N trade-offs $r_1, r_2, \ldots, r_N$ (i.e., a specific setting of the weights). These graphs have the same topology, since they are used for the tuning of the same set of parameters, but with different pheromone levels that encode knowledge for different trade-offs.

2) In each tuning iteration for a stage, one of the ACO graphs is selected and it suggests a new parameter configuration to try. To select an ACO graph, each ACO graph G_i is associated with the



((a)) ((b))

Figure 6.10: (a) Trade-offs with different weights for Pareto front exploration. (b) Calculation of hyper-volume.

history of reward improvement H_i, which is defined as:

$$H_i = \{r_{\text{sum}}^{(1)}, r_{\text{sum}}^{(2)}, \ldots, r_{\text{sum}}^{(t_i)}\}, \tag{6.8}$$

where t_i is the count of the times that G_i is selected. The term $r_{\text{sum}}^{(t_i)}$ is the summation of the reward improvement across all the N trade-offs brought by the t_i-th time of selecting G_i, which is given by:

$$r_{\text{sum}}^{(t_i)} = \sum_{1 \le j \le N} r_{\text{imp}(j)}^{(t_i)}, \tag{6.9}$$

where $r_{\text{imp}(j)}^{(t_i)}$ is the reward improvement (Line 20 in Algorithm 1) in terms of trade-off r_j brought by the t_i-th time of selecting G_i. We propose a multi-armed bandit-based selection rule as follows:

$$\mathcal{G} = \underset{1 \le i \le N}{\arg\max} \frac{\mu_i}{\sum_{1 \le j \le N} \mu_j} + k_r \sqrt{\frac{\log \sum_{1 \le j \le N} t_j}{t_i}}, \tag{6.10}$$

where $\mathcal{G}$ is the selected graph, μ_j is $\frac{1}{t_j}\left(r_{\text{sum}}^{(1)} + r_{\text{sum}}^{(2)} + \cdots + r_{\text{sum}}^{(t_j)}\right)$ and k_r is a hyper-parameter controlling the balance between choosing the graph that brings the most reward improvement and the one that is least frequently chosen.

3) Once an ACO graph is selected and the final QoR is obtained, the ACO graphs of the stage

are all updated based on (6.1) such that ϵ for G_i is calculated based on the corresponding trade-off r_i. It indicates that the information in every solution is used for the learning for all trade-offs concurrently.

6.7.2 Experimental Setup

We compared two methods with four sets of weights for (6.6): $(\frac{25}{26}, \frac{1}{26})$, $(\frac{5}{6}, \frac{1}{6})$, $(\frac{1}{2}, \frac{1}{2})$ and $(\frac{1}{6}, \frac{5}{6})$:

- **Multi-trade-off without concurrent learning**: FlowTuner was applied to each of four single-trade-off problems separately. 50 iterations for each trade-off, thus 200 iterations in total.
- **Multi-trade-off with concurrent learning**: FlowTuner with concurrent multi-trade-off learning was applied. The number of iterations was set to be 130[¶]. Note that, each iteration with or without concurrent learning takes similar runtime.

The performance of the Pareto front exploration was measured by the hyper-volume indicator [130]. Figure 6.10(b) illustrates its computation, where the reference point is the pair of the worst WNS and the worst area found. If the Pareto-front is pushed further forward, the hyper-volume indicator gets increased.

6.7.3 Results

Figure 6.11 gives a snapshot of the Pareto front exploration results. Each single-trade-off tuning discovers a small portion of the Pareto front while multi-trade-off tuning can discover an extensive range of the Pareto front. The average ratio of {hyper-volume of multi-trade-off tuning *with* concurrent learning in 130 iterations} against {hyper-volume of multi-trade-off tuning *without* concurrent learning in 200 iterations} across 5 testing designs is 0.999. It implies that the proposed concurrent multi-trade-off learning technique can push the Pareto front forward to the same amount with 35% less runtime.

[¶]We empirically found that, in our test circuits, 130 iterations of multi-trade-off tuning with concurrent learning achieved similar performance to 200 iterations of tuning without concurrent learning.

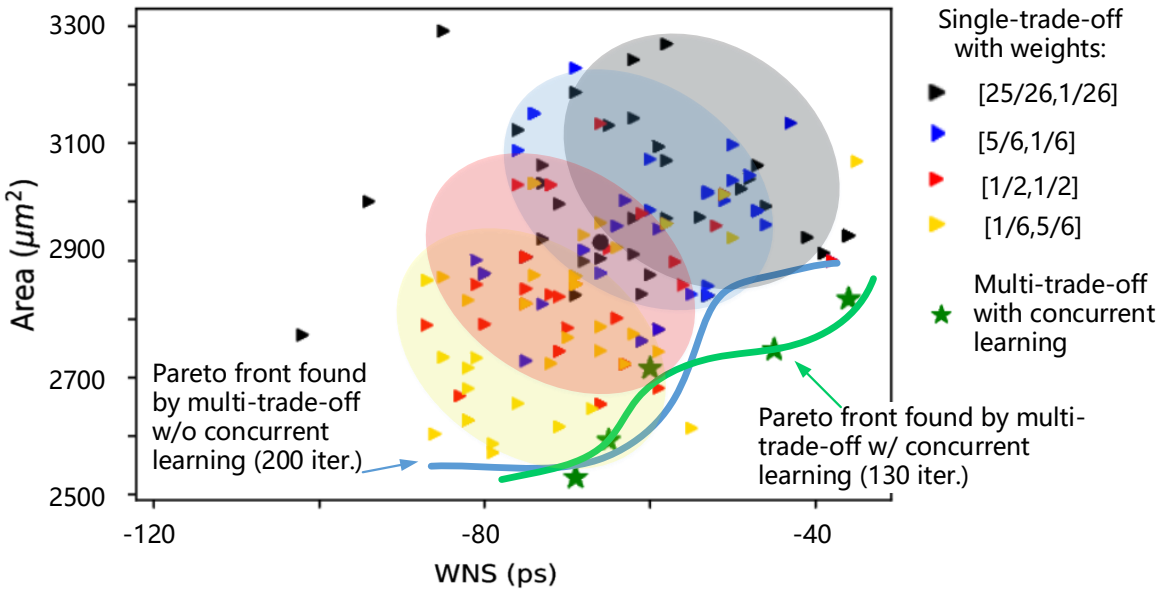


Figure 6.11: Snapshots of Pareto front exploration results.

6.8 Conclusion

In this chapter, we have presented a multi-stage EDA flow tuner, named FlowTuner, which utilizes both exploitation with knowledge transfer and exploration with a multi-stage cooperative co-evolutionary framework. The flow *jump-start* and *early-stop* techniques are actively deployed to address excessive runtime of EDA flow tuning. Novel concurrent multi-trade-off learning technique enables truly multi-objective tuning by facilitating efficient Pareto front exploration. Experiments through a flow using commercial tools have demonstrated that considerably better design outcomes are achieved in 50% shorter turnaround time compared against recent techniques.

7. CONCLUSIONS

This dissertation explores the opportunities of applying ML techniques to VLSI design automation with four example EDA tasks, i.e., DRC hotspot prediction, DRC heatmap prediction, routing-free crosstalk prediction and EDA flow parameter tuning.

For DRC hotspot prediction at cell placement stage, we propose a customized convolutional network architecture that can simultaneously take high resolution pin images and low resolution placement information as input features. This approach is evaluated on 12 industrial designs at 7nm process node and compared with three recent works. The results show that its prediction performance considerably outperforms the previous works and has fast feature extraction and inference time as no global routing information is required. This is an effort to customizing ML algorithms to address special needs in EDA tasks. Also, customized data augmentation techniques considering domain knowledge are proposed to alleviate the issue of “small data".

For the DRC heatmap prediction problem, to tackle the noisy DRC labels caused by non-deterministic parallel routing, we combine the strong modeling capability of deep neural network and the strength of stochastic models in coping with uncertainty. The stochastic distribution of DRVs in the layout is modelled by a Poisson point process and a Gaussian random field, based on which two novel techniques, the focal likelihood loss and the Gaussian random field layer, are proposed. Experiments results on industrial designs demonstrate that, these two techniques help achieve superior DRC heatmap prediction performance with noisy label data. This is an effort to address the issue of noisy data by integrating and customizing ML techniques. Given that the problem of noisy data caused by non-deterministic parallel computing is a general problem in applying ML techniques to EDA tasks, we hope our solution can be somehow instructive for addressing this issue.

We have proposed a routing-free ML-based crosstalk prediction framework. Given a placement, we extract net physical information, along with electrical, logical, and timing-related features. ML techniques are then employed to train crosstalk prediction models, which classify the

nets that are likely to have large coupling capacitance, crosstalk-induced noise, or incremental delay. Experimental validation on 12 benchmark circuits shows that the proposed method can classify more than 70% of crosstalk-critical nets after placement with a FPR of less than 2%. The computation speed is two orders of magnitude faster than a conventional method based on global routing. These results demonstrate that the proposed framework can serve as an accurate early-stage crosstalk evaluation engine that does not require routing information. In this example, various customized features are proposed to consider the joint effects of various types of information, i.e., physical, electrical and logical information. It is an effort to integrate domain knowledge with ML techniques.

For EDA flow parameter tuning, we have presented a multi-stage tuner, named FlowTuner, which utilizes both exploitation with knowledge transfer and exploration with a multi-stage cooperative co-evolutionary framework. The flow *jump-start* and *early-stop* techniques are actively deployed to address excessive runtime of EDA flow tuning. Experiments through a commercial tool flow have demonstrated that considerably better design outcomes are achieved in 50% shorter turnaround time compared against recent techniques. This is an effort to integrate evolutionary algorithms, stochastic methods and the branch-and-bound technique to handle complex EDA optimization problems.

This dissertation demonstrates great potential for applying ML techniques to VLSI design automation problems. Also, to handle special needs and issues in VLSI design automation tasks, we have presented efforts to customizing ML techniques by leveraging domain knowledge and have achieved promising results. We hope the dissertation can be instructive for future exploration in this direction.

REFERENCES

[1] A. B. Kahng, "Machine learning applications in physical design: Recent results and directions," in *Proceedings of International Symposium on Physical Design (ISPD)*, pp. 68–73, 2018.

[2] S. Chandrasekar, "Machine learning in digital IC design and EDA: Latest results and outlook," in *Proceedings of Design Automation Conference Tutorial (DAC Tutorial)*, 2019.

[3] Y. Liu, B. Cao, and H. Li, "Improving ant colony optimization algorithm with epsilon greedy and levy flight," *Complex & Intelligent Systems*, vol. 7, pp. 1711–1722, 2021.

[4] 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 International Symposium on Physical Design (ISPD)*, pp. 135–142, 2020.

[5] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of International Conference on Computer Vision (ICCV)*, pp. 2980–2988, 2017.

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

[7] R. Liang, J. Jung, H. Xiang, L. Reddy, L. Alexey, J. Hu, and G.-J. Nam, "Flowtuner: A multi-stage EDA flow tuner exploiting parameter knowledge transfer (in press)," in *Proceedings of International Conference On Computer Aided Design (ICCAD)*, 2021.

[8] Á. Fialho, R. Ros, M. Schoenauer, and M. Sebag, "Comparison-based adaptive strategy selection with bandits in differential evolution," in *Proceedings of International Conference on Parallel Problem Solving from Nature (PPSN)*, pp. 194–203, 2010.

[9] K. H. Yeap and H. Nisar, "Introductory chapter: VLSI," in *Very-Large-Scale Integration*, IntechOpen, 2018.

[10] C. Zhuo, B. Yu, and D. Gao, "Accelerating chip design with machine learning: From pre-silicon to post-silicon," in *Proceedings of International System-on-Chip Conference (SOCC)*, pp. 227–232, 2017.

[11] "International technology roadmap for semiconductors." `http://www.itrs2.net/itrs-reports.html`.

[12] J. H. Lee, J. Shin, and M. J. Realff, "Machine learning: Overview of the recent progresses and implications for the process systems engineering field," *Computers & Chemical Engineering*, vol. 114, pp. 111–121, 2018.

[13] "Machine learning for EDA - inside, outside and everywhere else." `https://semiwiki.com/eda/cadence/`.

[14] G. Huang, J. Hu, Y. He, J. Liu, M. Ma, Z. Shen, J. Wu, Y. Xu, H. Zhang, K. Zhong, *et al.*, "Machine learning for electronic design automation: A survey," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, pp. 1–46, 2021.

[15] B. K. Joardar, R. G. Kim, J. R. Doppa, P. P. Pande, D. Marculescu, and R. Marculescu, "Learning-based application-agnostic 3d noc design for heterogeneous manycore systems," *IEEE Transactions on Computers*, vol. 68, pp. 852–866, 2018.

[16] G. Mariani, G. Palermo, V. Zaccaria, and C. Silvano, "OSCAR: An optimization methodology exploiting spatial correlation in multicore design spaces," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, pp. 740–753, 2012.

[17] A. Deshwal, N. K. Jayakodi, B. K. Joardar, J. R. Doppa, and P. P. Pande, "MOOS: A multi-objective design space exploration and optimization framework for NoC enabled many-core systems," *ACM Transactions on Embedded Computing Systems*, vol. 18, pp. 1–23, 2019.

[18] G. Zhong, A. Prakash, S. Wang, Y. Liang, T. Mitra, and S. Niar, "Design space exploration of fpga-based accelerators with multi-level parallelism," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1141–1146, 2017.

[19] E. Zennaro, L. Servadei, K. Devarajegowda, and W. Ecker, "A machine learning approach for area prediction of hardware designs from abstract specifications," in *Proceedings of Euromicro Conference on Digital System Design (DSD)*, pp. 413–420, 2018.

[20] S. Dai, Y. Zhou, H. Zhang, E. Ustun, E. F. Young, and Z. Zhang, "Fast and accurate estimation of quality of results in high-level synthesis with machine learning," in *Proceedings of International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 129–132, 2018.

[21] R. G. Kim, J. R. Doppa, and P. P. Pande, "Machine learning for design space exploration and optimization of manycore systems," in *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pp. 1–6, 2018.

[22] K. O'Neal, M. Liu, H. Tang, A. Kalantar, K. DeRenard, and P. Brisk, "Hlspredict: Cross platform performance prediction for FPGA high-level synthesis," in *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2018.

[23] H. Chen and M. Shen, "A deep-reinforcement-learning-based scheduler for FPGA HLS," in *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2019.

[24] Z. Wang and B. C. Schafer, "Machine learning to set meta-heuristic specific parameters for high-level synthesis design space exploration," in *Proceedings of Design Automation Conference (DAC)*, pp. 1–6, 2020.

[25] E. Ustun, C. Deng, D. Pal, Z. Li, and Z. Zhang, "Accurate operation delay prediction for FPGA HLS using graph neural networks," in *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pp. 1–9, 2020.

[26] C. Yu, H. Xiao, and G. De Micheli, "Developing synthesis flows without human knowledge," in *Proceedings of Design Automation Conference (DAC)*, pp. 1–6, 2018.

[27] B. Shook, P. Bhansali, C. Kashyap, C. Amin, and S. Joshi, "MLParest: Machine learning based parasitic estimation for custom circuit design," in *Proceedings of Design Automation Conference (DAC)*, pp. 1–6, 2020.

[28] Y. Zhang, H. Ren, and B. Khailany, "GRANNITE: Graph neural network inference for transferable power estimation," in *Proceedings of Design Automation Conference (DAC)*, pp. 1–6, 2020.

[29] G. Pasandi, M. Peterson, M. Herrera, S. Nazarian, and M. Pedram, "Deep-PowerX: A deep learning-based framework for low-power approximate logic synthesis," in *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 73–78, 2020.

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

[31] A. Mirhoseini, H. Pham, Q. V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, and J. Dean, "Device placement optimization with reinforcement learning," in *Proceedings of International Conference on Machine Learning (ICML)*, pp. 2430–2439, 2017.

[32] Y.-C. Lu, J. Lee, A. Agnesina, K. Samadi, and S. K. Lim, "GAN-CTS: A generative adversarial framework for clock tree prediction and optimization," in *Proceedings of International Conference on Computer Design (ICCAD)*, pp. 1–8, 2019.

[33] T. Qu, Y. Lin, Z. Lu, Y. Su, and Y. Wei, "Asynchronous reinforcement learning framework for net order exploration in detailed routing," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1815–1820, 2021.

[34] 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 *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2018.

[35] 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," in *Proceedings of Design Automation Conference (DAC)*, pp. 1–6.

[36] 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 International Symposium on Physical Design (ISPD)*, pp. 57–64, 2020.

[37] C. Yu and Z. Zhang, "Painting on placement: Forecasting routing congestion using conditional generative adversarial nets," in *Proceedings of Design Automation Conference (DAC)*, pp. 1–6, 2019.

[38] M. B. Alawieh, W. Li, Y. Lin, L. Singhal, M. A. Iyer, and D. Z. Pan, "High-definition routing congestion prediction for large-scale FPGAs," in *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 26–31, 2020.

[39] C.-T. Ho and A. B. Kahng, "IncPIRD: Fast learning-based prediction of incremental IR drop," in *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 1–8, 2019.

[40] Z. Xie, H. Ren, B. Khailany, Y. Sheng, S. Santosh, J. Hu, and Y. Chen, "PowerNet: Transferable dynamic IR drop estimation via maximum convolutional neural network," in *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 13–18, 2020.

[41] H. Zhou, W. Jin, and S. X.-D. Tan, "GridNet: Fast data-driven EM-induced IR drop prediction and localized fixing for on-chip power grid networks," in *Proceedings of International Conference on Computer Aided Design (ICCAD)*, pp. 1–9, 2020.

[42] S. I. Ward, M.-C. Kim, N. Viswanathan, Z. Li, C. Alpert, E. E. Swartzlander Jr, and D. Z. Pan, "Keep it straight: Teaching placement how to better handle designs with datapaths," in *Proceedings of International Symposium on Physical Design (ISPD)*, pp. 79–86, 2012.

[43] S. Ward, D. Ding, and D. Z. Pan, "PADE: A high-performance placer with automatic datapath extraction and evaluation through high-dimensional data learning," in *Proceedings of Design Automation Conference (DAC)*, pp. 756–761, 2012.

[44] E. C. Barboza, N. Shukla, Y. Chen, and J. Hu, "Machine learning-based pre-routing timing prediction with reduced pessimism," in *Proceedings of Design Automation Conference (DAC)*, pp. 1–6, 2019.

[45] D. Ding, J.-R. Gao, K. Yuan, and D. Z. Pan, "AENEID: a generic lithography-friendly detailed router based on post-RET data learning and hotspot detection," in *Proceedings of Design Automation Conference (DAC)*, pp. 795–800, 2011.

[46] S. Li and B. Jacob, "Statistical DRAM modeling," in *Proceedings of the International Symposium on Memory Systems (MEMSYS)*, pp. 521–530, 2019.

[47] D. Lee and A. Gerstlauer, "Learning-based, fine-grain power modeling of system-level hardware IPs," *ACM Transactions on Design Automation of Electronic Systems*, vol. 23, pp. 1–25, 2018.

[48] W. Chen, K.-K. Hsieh, L.-C. Wang, and J. Bhadra, "Data-driven test plan augmentation for platform verification," *IEEE Design & Test*, vol. 34, pp. 23–29, 2017.

[49] D. Kim, P. Kang, S. Cho, H.-j. Lee, and S. Doh, "Machine learning-based novelty detection for faulty wafer detection in semiconductor manufacturing," *Expert Systems with Applications*, vol. 39, pp. 4075–4083, 2012.

[50] A. DeOrio, Q. Li, M. Burgess, and V. Bertacco, "Machine learning-based anomaly detection for post-silicon bug diagnosis," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 491–496, 2013.

[51] Y. Ma, H. Ren, B. Khailany, H. Sikka, L. Luo, K. Natarajan, and B. Yu, "High performance graph convolutional networks with applications in testability analysis," in *Proceedings of Design Automation Conference (DAC)*, pp. 1–6, 2019.

[52] X. Sun, K. Chakrabarty, R. Huang, Y. Chen, B. Zhao, H. Cao, Y. Han, X. Liang, and L. Jiang, "System-level hardware failure prediction using deep learning," in *Proceedings of Design Automation Conference (DAC)*, pp. 1–6, 2019.

[53] H. Wang, W. Cai, J. Li, and K. He, "Exploring graphical models with bayesian learning and MCMC for failure diagnosis," in *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 151–156, 2020.

[54] B. Mammo, M. Furia, V. Bertacco, S. Mahlke, and D. S. Khudia, "Bugmd: Automatic mismatch diagnosis for bug triaging," in *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, 2016.

[55] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi, *et al.*, "A graph placement methodology for fast chip design," *Nature*, vol. 594, pp. 207–212, 2021.

[56] A. B. Kahng, S. Kumar, and T. Shah, "A no-human-in-the-loop methodology toward optimal utilization of EDA tools and flows," in *Proceedings of Design Automation Conference work in progress poster (DAC WIP)*, pp. 1–6, 2018.

[57] J. Kwon, M. M. Ziegler, and L. P. Carloni, "A learning-based recommender system for autotuning design fiows of industrial high-performance processors," in *Proceedings of Design Automation Conference (DAC)*, pp. 1–6, 2019.

[58] Z. Xie, G.-Q. Fang, Y.-H. Huang, H. Ren, Y. Zhang, B. Khailany, S.-Y. Fang, J. Hu, Y. Chen, and E. C. Barboza, "FIST: A feature-importance sampling and tree-based method for automatic design flow parameter tuning," in *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 19–25, 2020.

[59] S. Liu, Q. Sun, P. Liao, Y. Lin, and B. Yu, "Global placement with deep learning-enabled explicit routability optimization," *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1821–1824, 2021.

[60] T. M. Mitchell, "Does machine learning really work?," *AI magazine*, vol. 18, pp. 11–11, 1997.

[61] P. Domingos, *The master algorithm: How the quest for the ultimate learning machine will remake our world*. Basic Books, 2015.

[62] T. M. Oshiro, P. S. Perez, and J. A. Baranauskas, "How many trees in a random forest?," in *Proceedings of workshop on machine learning and data mining in pattern recognition (MLDM)*, pp. 154–168, 2012.

[63] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of International Conference on Knowledge Discovery and Data Mining (DMKD)*, pp. 785–794, 2016.

[64] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *Proceedings of International Conference on Engineering and Technology (ICET)*, pp. 1–6, 2017.

[65] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440, 2015.

[66] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE computational intelligence magazine*, vol. 1, pp. 28–39, 2006.

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

[68] J. Lou, S. Thakur, S. Krishnamoorthy, and H. S. Sheng, "Estimating routing congestion using probabilistic analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, pp. 32–41, 2002.

[69] P. Spindler and F. M. Johannes, "Fast and accurate routing demand estimation for efficient routability-driven placement," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1226–1231, 2007.

[70] T. Taghavi, Z. Li, C. Alpert, G.-J. Nam, A. Huber, and S. Ramji, "New placement prediction and mitigation techniques for local routing congestion," in *Proceedings o International Conference on Computer-Aided Design (ICCAD)*, pp. 621–624, 2010.

[71] 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 International Symposium on Physical Design (ISPD)*, pp. 15–21, 2017.

[72] A. F. Tabrizi, L. Rakai, N. K. Darav, I. Bustany, L. Behjat, S. Xu, and A. Kennings, "A machine learning framework to identify detailed routing short violations from a placed netlist," in *Proceedings of Design Automation Conference (DAC)*, pp. 1–6, 2018.

[73] J. Seo, J. Jung, S. Kim, and Y. Shin, "Pin accessibility-driven cell layout redesign and placement optimization," in *Proceedings of Design Automation Conference (DAC)*, pp. 1–6, 2017.

[74] 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," in *Proceedings of Design Automation Conference (DAC)*, pp. 1–6, 2019.

[75] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proceedings of International Conference on Medical Image Computing and Computer-assisted Intervention (MICCAI)*, pp. 234–241, 2015.

[76] W. L. Briggs, S. F. McCormick, *et al.*, *A multigrid tutorial*, vol. 72. Siam, 2000.

[77] J. Cong and J. R. Shinnerl, *Multilevel optimization in VLSI CAD*, vol. 14. Springer Science & Business Media, 2013.

[78] W. Zeng, A. Davoodi, and Y. H. Hu, "Design rule violation hotspot prediction based on neural network ensembles," *arXiv preprint arXiv:1811.04151*, 2018.

[79] Y.-H. Huang, Z. Xie, G.-Q. Fang, T.-C. Yu, H. Ren, S.-Y. Fang, Y. Chen, and J. Hu, "Routability-driven macro placement with embedded CNN-based prediction model," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 180–185, 2019.

[80] "Tips and tricks to make gans work." `https://github.com/soumith/ganhacks`.

[81] M. Liu, X. Zhang, Z. Chen, X. Wang, and T. Yang, "Fast stochastic AUC maximization with O (1/n)-convergence rate," in *Proceedings of International Conference on Machine Learning (ICML)*, pp. 3195–3203, 2018.

[82] J. Davis and M. Goadrich, "The relationship between precision-recall and ROC curves," in *Proceedings of International Conference on Machine Learning (ICML)*, pp. 233–240, 2006.

[83] R. Islam and M. A. Shahjalal, "Late breaking results: Predicting DRC violations using ensemble random forest algorithm," in *Proceedings of Design Automation Conference (DAC)*, pp. 1–2, 2019.

[84] L.-C. Chen, C.-C. Huang, Y.-L. Chang, and H.-M. Chen, "A learning-based methodology for routability prediction in placement," in *Proceedings of International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pp. 1–4, 2018.

[85] C. Li, M. Xie, C.-K. Koh, J. Cong, and P. H. Madden, "Routability-driven placement and white space allocation," *IEEE Transactions on Computer-aided design of Integrated Circuits and Systems*, vol. 26, pp. 858–871, 2007.

[86] Q. Zhou, X. Wang, Z. Qi, Z. Chen, Q. Zhou, and Y. Cai, "An accurate detailed routing routability prediction model in placement," in *Proceedings of Asia Symposium on Quality Electronic Design (ASQED)*, pp. 119–122, 2015.

[87] J. Moller and R. P. Waagepetersen, *Statistical inference and simulation for spatial point processes*. CRC Press, 2003.

[88] J. Møller, A. R. Syversveen, and R. P. Waagepetersen, "Log gaussian cox processes," *Scandinavian journal of statistics*, pp. 451–482, 1998.

[89] S. Z. Li, *Markov random field modeling in image analysis*. Springer Science & Business Media, 2009.

[90] A. Vittal and M. Marek-Sadowska, "Crosstalk reduction for VLSI," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, pp. 290–298, 1997.

[91] K. Rahmat, J. Neves, and J.-F. Lee, "Methods for calculating coupling noise in early design: a comparative analysis," in *Proceedings of International Conference on Computer Design (ICCAD)*, pp. 76–81, 1998.

[92] H. Ren, D. Pan, and P. G. Villarubia, "True crosstalk aware incremental placement with noise map," in *Proceedings of International Conference on Computer Aided Design (ICCAD)*, pp. 402–409, 2004.

[93] H. Zhou and M. D. F. Wong, "Global routing with crosstalk constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, pp. 1683–1688, 1999.

[94] P. N. Parakh and R. B. Brown, "Crosstalk constrained global route embedding," in *Proceedings of the International Symposium on Physical Design (ISPD)*, pp. 201–206, 1999.

[95] J. Lou and W. Chen, "Crosstalk-aware placement," *IEEE Design & Test of Computers*, vol. 21, pp. 24–32, 2004.

[96] D. Wu, J. Hu, R. Mahapatra, and M. Zhao, "Layer assignment for crosstalk risk minimization," in *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 159–162, 2004.

[97] H.-P. Tseng, L. Scheffer, and C. Sechen, "Timing-and crosstalk-driven area routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, pp. 528–544, 2001.

[98] M. R. Becer, D. Blaauw, I. Algor, R. Panda, C. Oh, V. Zolotov, and I. N. Hajj, "Postroute gate sizing for crosstalk noise reduction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, pp. 1670–1677, 2004.

[99] C. J. Alpert, A. Devgan, and S. T. Quay, "Buffer insertion for noise and delay optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, pp. 1633–1645, 1999.

[100] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of Conference and Workshop on Neural Information Processing Systems (NIPS)*, pp. 1024–1034, 2017.

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

[102] M. Kuhlmann and S. S. Sapatnekar, "Exact and efficient crosstalk estimation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, pp. 858–866, 2001.

[103] L. H. Chen and M. Marek-Sadowska, "Incremental delay change due to crosstalk noise," in *Proceedings of International Symposium on Physical Design (ISPD)*, pp. 120–125, 2002.

[104] K. Agarwal, D. Sylvester, and D. Blaauw, "Modeling and analysis of crosstalk noise in coupled RLC interconnects," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, pp. 892–901, 2006.

[105] M. R. Becer, D. Blaauw, I. N. Hajj, and R. Panda, "Early probabilistic noise estimation for capacitively coupled interconnects," in *Proceedings of International Workshop on System-level Interconnect Prediction (SLIP)*, pp. 77–83, 2002.

[106] F.-Y. Fan, H.-M. Chen, and I. Liu, "Technology mapping with crosstalk noise avoidance," in *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 319–324, 2010.

[107] H. Sankaran and S. Katkoori, "Simultaneous scheduling, allocation, binding, re-ordering, and encoding for crosstalk pattern minimization during high–level synthesis," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 19, pp. 217–226, 2009.

[108] C. Duan, A. Tirumala, and S. P. Khatri, "Analysis and avoidance of cross-talk in on-chip buses," in *Proceedings of International Symposium on High Performance Interconnects (ISHPI)*, pp. 133–138, 2001.

[109] A. B. Kahng, M. Luo, and S. Nath, "SI for free: machine learning of interconnect coupling delay and transition effects," in *Proceedings of International Workshop on System Level Interconnect Prediction (SLIP)*, pp. 1–8, 2015.

[110] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[111] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proceedings of Conference on Neural Information Processing Systems (NIPS)*, pp. 8024–8035, 2019.

[112] T. M. Khoshgoftaar, M. Golawala, and J. Van Hulse, "An empirical study of learning from imbalanced data using random forest," in *Proceedings of International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 310–317, 2007.

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

[114] C. Albrecht, "Iwls 2005 benchmarks," in *Proceedings of International Workshop for Logic Synthesis (IWLS)*, pp. 9–9, 2005.

[115] 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.

[116] G. Menardi and N. Torelli, "Training and assessing classification rules with imbalanced data," *Data Mining and Knowledge Discovery*, vol. 28, pp. 92–122, 2014.

[117] M. M. Ziegler, H.-Y. Liu, G. Gristede, B. Owens, R. Nigaglioni, and L. P. Carloni, "A synthesis-parameter tuning system for autonomous design-space exploration," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1148–1151, 2016.

[118] H.-Y. Liu and L. P. Carloni, "On learning-based methods for design-space exploration with high-level synthesis," in *Proceedings of Design Automation Conference (DAC)*, pp. 1–7, 2013.

[119] Y. Ma, Z. Yu, and B. Yu, "CAD tool design space exploration via bayesian optimization," in *Proceedings of Workshop on Machine Learning for CAD (MLCAD)*, pp. 1–6, 2019.

[120] C. Xu, G. Liu, R. Zhao, S. Yang, G. Luo, and Z. Zhang, "A parallel bandit-based approach for autotuning FPGA compilation," in *Proceedings of International Symposium on Field-Programmable Gate Arrays (FPGA)*, pp. 157–166, 2017.

[121] E. Ustun, S. Xiang, J. Gui, C. Yu, and Z. Zhang, "Lamda: Learning-assisted multi-stage autotuning for FPGA design closure," in *Proceedings of International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 74–77, 2019.

[122] M. K. Papamichael, P. Milder, and J. C. Hoe, "Nautilus: Fast automated IP design space search using guided genetic algorithms," in *Proceedings of Design Automation Conference (DAC)*, pp. 1–6, 2015.

[123] P. Meng, A. Althoff, Q. Gautier, and R. Kastner, "Adaptive threshold non-Pareto elimination: Re-thinking machine learning for system level design space exploration on FPGAs," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 918–923, 2016.

[124] A. Agnesina, K. Chang, and S. K. Lim, "VLSI placement parameter optimization using deep reinforcement learning," in *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pp. 1–9, 2020.

[125] X. Ma, X. Li, Q. Zhang, K. Tang, Z. Liang, W. Xie, and Z. Zhu, "A survey on cooperative co-evolutionary algorithms," *IEEE Transactions on evolutionary computation*, vol. 23, pp. 421–441, 2018.

[126] M. Munetomo and D. E. Goldberg, "Linkage identification by non-monotonicity detection for overlapping functions," *Evolutionary computation*, vol. 7, pp. 377–398, 1999.

[127] "Bayesian Optimization." `https://github.com/fmfn/BayesianOptimization`.

[128] Q. Zhang and H. Li, "MOEA/D: A multi-objective evolutionary algorithm based on decomposition," *IEEE Transactions on evolutionary computation*, vol. 11, pp. 712–731, 2007.

[129] M. T. Emmerich and A. H. Deutz, "A tutorial on multiobjective optimization: fundamentals and evolutionary methods," *Natural computing*, vol. 17, pp. 585–609, 2018.

[130] J. Bader and E. Zitzler, "HypE: An algorithm for fast hypervolume-based many-objective optimization," *Evolutionary computation*, vol. 19, pp. 45–76, 2011.