# Methodologies for Efficient Hardware Design Automation and Hardware-friendly Learning

**MA, Yuzhe**

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Doctor of Philosophy
in
Computer Science and Engineering

The Chinese University of Hong Kong
June 2020

<u>Thesis Assessment Committee</u>

Professor LEE Ho Man (Chair)

Professor YU Bei (Thesis Supervisor)

Professor YOUNG Fung Yu (Committee Member)

Professor PAN David (External Examiner)

# Abstract

Modern silicon systems are constituted by hardware (e.g., integrated circuits) and software (e.g., applications). It is essential to ensure prominent hardware design and desired coordination from software to achieve high efficacy of the system. On the hardware side, the complexity of hardware design is increasing as the technology node keeps shrinking. Consequently, more design constraints are imposed and lead to a long design cycle as well as a performance bottleneck, which calls for efficient design automation methodologies. On the software side, the rapid development of deep learning (DL) has catalyzed many innovative applications for intelligent systems. However, typical DL models require a significant amount of resources on storage and computation, which hinders the deployment on resource-constrained hardware. Therefore, methodologies for hardware-friendly learning is required to ensure the efficacy of the system.

This dissertation attempts to present our research on a set of novel methodologies for efficient hardware design automation and hardware-friendly learning, and demonstrates the effectiveness in different stages covering a wide range along the design flow of silicon systems. Our research includes graph learning in testability optimization, active leaning in performance optimization, unified manufacturability optimization, and hardware-friendly neural network optimization.

Firstly, active learning methodology is studied for performance optimization. Due to the inevitable optimality gap between the architectural level and physical design level, traditional iterative tuning may cause an extremely long design cycle when the design space is huge. We present an active learning methodology for cross-layer optimization, which is able to explore the design space and obtain the Pareto-optimal designs efficiently while maintaining a low data labeling cost.

Secondly, the graph learning methodology is investigated for testability optimization. Test point insertion is a critical step in design-for-testing. Thanks to the high scalability of the data-driven learning techniques, they have become promising alternatives to traditional heuristic methods and simulation methods. To tackle the irregular structures of the netlists, a graph learning technique is explored such that the powerful learning techniques can be smoothly leveraged, which fuses the conventional learning approaches with typical circuit design problems. Besides, we propose a specialized workflow for test point insertion based on the graph learning model.

Thirdly, a unified manufacturability optimization framework is presented. Layout decomposition and mask optimization are two key stages in advanced technology nodes. However, the conventional two-stage flow (i.e., layout decomposition followed by mask optimization) cannot achieve good printability on their own. To tackle this problem, a unified optimization framework is studied, which seamlessly integrates these two stages. Combining the two processes together leads to a larger solution space and can obtain higher quality masks.

For hardware-friendly neural network optimization, a unified framework to compress any well-trained neural networks is firstly explored, which leverages both low-rankness and sparsity for efficient calculation and can be applied in general scenario like supervised classification and regression tasks. In additional to the general compression scheme, specifically designed schemes are also proposed to fully explore the room of compression under special scenarios. A specific pruning strategy is investigated for partial domain adaptation which is a special application of deep neural networks.

The effectiveness of proposed design methodologies is demonstrated on extensive experiments on industrial benchmarks and widely used open benchmarks. These methodologies are capable of enhancing and accelerating hardware design automation, and enabling hardware-friendly learning.

# 摘要

現代數字系統主要是由硬件（例如集成電路）和軟件（例如應用程式）組成。為了達到系統的高效能，保證精良的硬件設計以及了理想的軟件協調至關重要。在硬件方面，隨著技術節點的縮小，設計複雜度日漸上升。更多的設計約束導致了冗長的設計週期以及性能瓶頸，因此需要更加高效的設計自動化方法學來帮助达到目标。在軟件方面，深度學習技術的發展催生了很多用於智能系統的創新應用。然而，典型的深度學習模型需要大量的存儲和運算資源，因此也限制了此類應用在資源制約型硬件上面的部署。因此，高效能系統也需要能促進硬件友好型機器學習技術的方法學。

本論文將展示多種新型的高效的硬件設計自動化方法和硬件友好型深度學習方法的研究，並且顯示出這些方法在系統設計流程中多個不同階段的有效性。研究內容包括用於可測試性優化的圖學習方法，用於性能優化的主動學習方法，用於可測試性設計的聯合優化方法以及硬件友好型的神經網絡優化方法。

首先，我們研究了用於性能優化的主動學習技術。由於在電路設計的架構設計層和物理實現層有著不可避免的最優間隙，在設計空間非常大的時候傳統迭代調試的方法會導致很長的設計週期。為了解決這個問題，我們展示了一種可以實現跨層優化的主動學習方法。此方法能高效的探索設計空間以獲得帕累托最優的設計點。另外，此方法還能有效的控制數據標註的代價。

其次，我們研究了應用於可測試性優化的圖學習方法。測試點插入是可測試性設

計中的關鍵步驟。考慮到機器學習方法的強擴展性，這類方法逐漸替代了傳統基於電路模擬算法和啟發式算法。為了處理電路網表的非規則結構，我們研究了一種圖學習方法。這種方法融合了通用的機器學習和電路設計問題。基於這種方法所得到的圖學習模型，我們提出了一種特定的測試點插入的設計流程。

第三，我們展示了一種用於可製造性設計的統一優化方法。在可製造性設計中，版圖分解和掩模優化是兩個最核心的步驟。然而，僅依賴通用的兩階段流程（版圖分解–掩模優化）不能得到良好的製造結果。為了解決這個問題，我們研究出一種統一可製造性優化框架來無縫集成這兩個步驟。將兩個步驟合併之後得到了更大的解空間並且也得到了更高質量的掩模。

對於硬件友好型神經網路優化，首先我們研究了一種統一框架結合了低秩性和稀疏性來實現對預訓練網路的壓縮和加速。這樣的框架能在大量的常見場景中使用，例如監督式分類和回歸任務。除了通用的壓縮策略之外，我們還研究了訂製的壓縮策略以充分探索在特殊應用下的壓縮空間。我們還研究了一種用於局部域適應特殊場景下的壓縮方法來移除預訓練中模型中與目標任務不相關的參數。

我們在工業級測試基準和廣泛使用的公開測試基準上進行了大量的實驗，實驗結果驗證了我們所提出的設計方法學的有效性，顯示出這一系列方法能夠增強以及加速硬體設計自動化以及促進硬體友好型深度學習的實現。

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Hardware and software are two main components that constitute the prosperous information technology industry. The very large scale integration (VLSI) has become one of the most representative technologies, which was under rapid evolvement in the last few decades. Consequently, today's integrated-circuits (ICs) are becoming smaller, more powerful, and more power-efficient, which form the hardware foundation of all the digital applications. Complementary to the hardware, software that runs on hardware provides all kinds of services such that the entire system can operate as desired. For instance, Google's image retrieval system leverages deep learning software running on the hardware named Tensor Processing Unit to provide services for users.

In order to design and optimize a VLSI circuit, there has been a conventional design flow for modern circuits, which consists of several typical stages, as shown in **??**. It starts from system level specification and architectural design. Then the system is implemented with hardware languages. Then logic synthesis converts implementation to gate level netlist. Physical design realizes circuit components into a layout with positions and sizes of gates and wires. Then verification and design rule checks need to be conducted, and then chips can be fabricated in a foundry. When designing the software, it is vital to be compatible with the hardware to achieve the maximum benefit. Specifically, the hardware resource constraints should be taken into consideration, such as power and storage constraints. With the outstanding achievements made by machine learning and deep learning, a trending research direction is the hardware-friendly learning to well coordinate the algorithms and the hardware.

## 1.1 Challenges

In spite of the great achievements that have been made through the design flow, there are also various emerging challenges. The issues can be categorized into three parts.

Firstly, the complexity of hardware design is increasing as the technology node keeps shrinking. Thus, more constraints are imposed and the problem size is enlarging as well. What's worse, many problems in the design flow are NP-hard, leading to performance bottleneck and severe runtime overhead in certain design stages. Therefore, novel effective methodologies for improving the efficiency are in demand.

Secondly, the conventional long design flow is separated into multiple stages and each stage is tackled separately. Due to the misalignment among the objectives of different stages, the performance gap is inevitable, which requires enormous iterations of engineering tuning to explore the design space. Considering the design space is so large that it is extremely time-consuming to be explored exhaustively, methodologies for efficient design space exploration and unified optimization across multiple stages are very promising to further enhance and accelerate the hardware design automation.

Thirdly, the development of deep learning has catalyzed many innovative applications in various kinds of real-world software. However, large model size and inten-

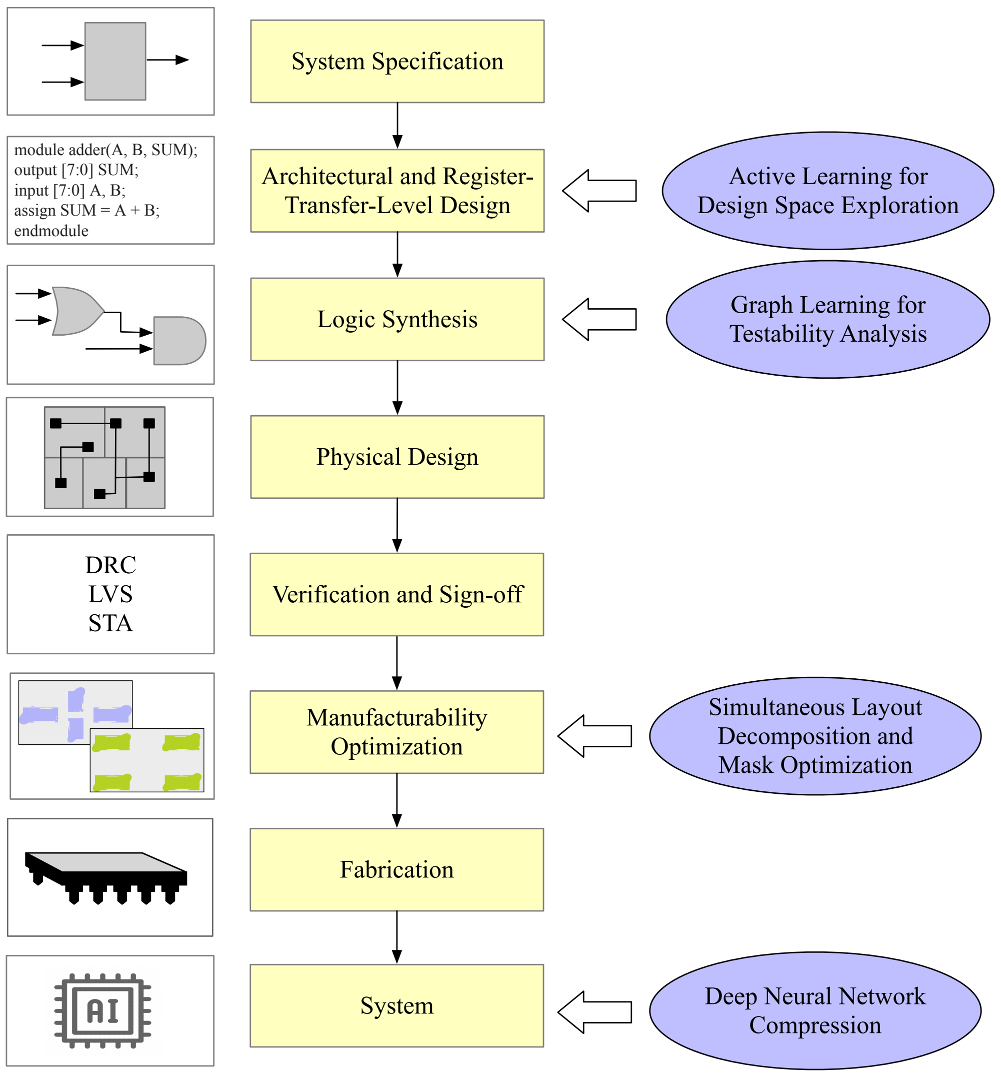

module adder(A, B, SUM);
output [7:0] SUM;
input [7:0] A, B;
assign SUM = A + B;
endmodule
System Specification
Architectural and Register-Transfer-Level Design
Active Learning for Design Space Exploration
Logic Synthesis
Graph Learning for Testability Analysis
Physical Design
DRC
LVS
STA
Verification and Sign-off
Manufacturability Optimization
Simultaneous Layout Decomposition and Mask Optimization
Fabrication
AI
System
Deep Neural Network Compression

Figure 1.1: Conventional design flow for a digital system, and the corresponding stages of the proposed techniques.

sive computations hinder the broad deployment of deep neural networks on resource-constrained hardware. In order to fully exploit the capability of deep learning, designing hardware-friendly learning models become essential. Particularly, since deep learning has covered a wide range of applications and scenarios, the methodologies of enabling hardware-friendly learning should contain general techniques as well as tackle special scenarios.

## 1.2 Thesis Overview

This thesis attempts to enhance and accelerate both hardware design automation and hardware-friendly learning. Several design stages are investigated, which are shown in Figure 1.1.

Chapter 3 investigates the design space exploration problem for modern designs. The purpose is to obtain the desired design in an enormous design space effectively and efficiently. A Pareto frontier-driven active learning method is proposed to explore the design space of 64-bit adders, and achieve good trade-offs among power, area, and delay, while maintaining a low data labeling cost for learning.

Chapter 4 proposes a graph learning-based methodology for testability optimization, which focuses on the test point insertion problem. In order to leverage the capability of the data-driven approaches, a high-performance graph convolutional network (GCN) model is proposed for the purpose of processing irregular graph representations of logic circuits, which is further integrated into a test point insertion flow and achieve comparable performance to commercial testability analysis tools.

Chapter 5 presents a unified optimization framework for manufacturability optimization, which seamlessly integrates layout decomposition and mask optimization. To tackle the inconsistency of the objectives of these two stages, a unified mathematical formulation is proposed, and it is solved with a set of numerical and combinatorial techniques.

Chapter 6 focuses on hardware-friendly deep learning which aims to coordinate the executed algorithms with hardware to achieve good trade-off among performance, power consumption, and throughput. Two aspects are studied: (1) A unified approximation framework for accelerating and compressing deep neural network is firstly introduced, which is a general technique and can be applied to most of the common deep models; (2) A pruning methodology is proposed for unsupervised partial domain adaptation, which is a special scenario in deep learning applications.

# Chapter 2

# Literature Review

## 2.1 Design Space Exploration and Active Learning

Very large scale integration (VLSI) design methodologies have developed about 50 years, from manually-crafted design to computer-aided design (CAD) with increasingly higher levels of the design specification. With the aggressive scaling of technology nodes, design complexity increases dramatically and leads to a huge design space for a modern design. As a result, efficient design space exploration (DSE) has emerged as a promising solution to tackle the exponentially increasing size of the design space of microprocessors and the consequent time-consuming synthesis runs, among which the learning-based approached are used intensively. However, the data labeling cost should always be considered in the machine learning application, especially in IC design. Active learning is a machine learning algorithm that can obtain better performance with fewer training data, provided that the data is selected based on what the algorithm learns. This paradigm becomes more and more important in various scenarios where there is a large amount of quantity of unlabeled data and the labels are very few due to the time-consuming and expensive labeling process. This section reviews the literature in both DSE and active learning.

There are a variety of algorithms investigated to explore design space in different design scenarios, ranging from high level synthesis to layout generation. Liu *et al.* [9] presented a random forest-based learning model in high level synthesis, which can find an approximated Pareto-optimal designs effectively. Meng *et al.* [10] proposed a random forest-based method for Pareto frontier exploration, where non-Pareto-optimal designs are carefully eliminated through an adaptive strategy. Multiple predictions can be obtained through the random forest, which can be used for estimating the uncertainty. Palermo *et al.* [11] deployed both linear regression and artificial neural networks for multiprocessor systems-on-chip design. Apart from tuning the design parameters, exploring the parameters in EDA tools has also been studied. FIST [12] leveraged a boosting model and a feature importance identification scheme for automatic parameter tuning in EDA tools. IBM built an entire workflow, SynTunSys, to assist in reducing the design cycle in advanced technology nodes [13, 14, 15].

In addition, DSE for analog circuits design has also been heavily explored recently, in which Bayesian optimization is broadly applied. Lyu *et al.* [16] proposed a multi-objective Bayesian optimization framework for analog circuits synthesis. Despite its superior efficiency to traditional design methodologies, the number of samples each iteration limits the overall improvement. To address that, a batch Bayesian optimization framework ensembling multiple acquisition functions was proposed [17], which samples multiple points by solving a multi-objective acquisition function. Gaussian process [18] is a widely used surrogate model in Bayesian optimization. However, considering that complexity of training a Gaussian process model is $\mathcal{O}(n^3)$, easing the training overhead and sample efficiency could further accelerate the process. A multi-fidelity framework [19] was proposed to leverage more data with low labeling cost and less data with high data labeling cost to build the model. The neural network is applied as an explicit kernel to approximate the implicit kernel representation [20] in the traditional Gaussian Process, such that the training time scales linearly with the number of training samples.

An active learning model requires queries in the data pool in order to get specific unlabeled data points to be labeled, which in return can be applied to further train the model. Therefore, evaluating the informativeness of unlabeled instances is the most critical problem in an active learning algorithm.

Many approaches have been proposed to formulate the query schemes, which can be categorized into three classes. The most commonly applied scheme is query based on uncertainty, which is to select a data point with the least confidence on how to label. Typically it is utilized in classification problem [21, 22, 23, 24]. In the case of regression problems, the output is a continuous value instead of a discrete label. Thus the uncertainty can be characterized by leveraging the variance of the output [25, 26, 27].

The second query scheme is the query-by-committee (QBC) algorithm [28], which involves maintaining a committee of different models that are all trained on the same data set. Thus, each model in this committee would generate one label for every query candidate. The one which receives the most disparate labels is determined to be the most informative and then be queried for its label [29, 30, 31].

Another algorithm for selecting the query point is the expected-model-change (EMC) which selects the point with the greatest impact on the current model given its new label. A representative framework of EMC is the expected-gradient-length (EGL) [32]. The intuition is that the instance leading to the greatest change in gradient of the loss function with respect to the model parameters is more likely to be more informative. Generally, the EGL method is applicable to learning problems which use gradient-based methods for training, and was investigated is several works [32, 33, 34].

## 2.2 Testability Analysis and Graph Learning

As the technology node scales down, there are billions of transistors on a single die, which is much more prone to defects than ever. Therefore, circuits testing for manufacturing defects is of great importance in the production cycle of ICs since it affects the reliability and development cost. In order to test a circuit, firstly a set of test patterns needs to be generated, which is also called test set. Each pattern is a binary sequence (vector) whose length is equal to the number of circuit inputs. Therefore, the size of the input vector space is $2^n$ for a circuit with $n$ inputs, which grows exponentially with $n$. Generating a test set with sufficient test patterns relies on automatic test patterns generation (ATPG) tools that analyze the circuit netlist and produce corresponding representative test patterns. However, ATPG is prone to runtime overhead due to the poor testability of a circuit if the testability is not well-considered in the design stage, which motivates the research on testability analysis and design-for-testing (DFT).

Test point insertion (TPI) is a broadly used approach in DFT to modify a circuit and improve its testability, which involves adding extra control points (CPs) or observation points (OPs) to the circuit. CPs can be used for setting signal lines to desired logic values, while OPs are added as scan cells to make a node observable. There are several issues that needed to be considered when performing TPI. On one hand, the optimal test point placement problem is NP-complete [35]. Numerous TPI methods have been proposed to investigate the efficiency and performance of TPI. Based on the runtime, these methods can be categorized into exact fault simulation [36], approximate measurements [37] and simulation-based methods [38]. On the other hand, inserting test points may degrade the performance of a design in terms of area, power, and timing. The ultimate goal of TPI is to achieve high fault coverage with less performance degradation. Previous works explored beneficial trade-offs between testability improvement and performance degradation [39, 40, 41, 42, 43], among which CPs insertion is considered in [42] and OPs insertion is considered in [43]. Touba *et al.* [40] consider inserting both CPs and OPs.

The data-driven approaches are promising solutions for TPI. However, since a netlist or circuit is usually modeled as a graph which is an irregular structure rather than a regular one like a sequence or an image. Graph is a fundamental object in many fields, which is a mathematical structure that models pairwise relationships among different items. However, typical learning-based approaches are not directly applicable to graph-related problems due to the irregular structures. Graph learning is a new approach to machine learning with a wide range of applications [44]. One advantage is the graph structure itself can reveal relevant information. Before performing a certain task, representation of a node or graph should be obtained first, which is known as embedding and can be fed to downstream models. Previous approaches exploring node embedding problems can be classified into two categories. The first class of approaches is based on heuristics to encode the structural information [45]. A more recent approach is data-driven, which learns node embeddings automatically [46, 2, 47].

Graph convolution networks are one of these data-driven approaches. Within these data-driven approaches, they can be further classified into transductive and inductive. Transductive approaches directly optimize the embedding for each node, thus they require all nodes to be present during training, and hence cannot generalize to unseen graphs [46]. Inductive approaches generate node embeddings through learning a set of functions to aggregate the structural information and node attributes, which make the learned model independent from training graphs. Therefore, inductive models can be applied to unseen data [2, 47].

Unlike conventional graph learning tasks, graph learning for EDA problems is prone to runtime overhead considering that the scale of circuits keeps soaring. Similar to conventional CNNs, the most time-consuming process in the computation of a GCN is the embedding generation. To tackle the issue of scalability, several attempts have been made for efficient graph representation learning. It is pointed out that the inefficiency might be caused by duplicated computation under the GraphSAGE-like framework [47]. To address this, PinSAGE [47] is proposed to select important neighbors by random walk instead of aggregating all the neighbors, and a MapReduce pipeline is leveraged for maximizing the inference throughput of a trained model. Recently, GraphZoom [48] is proposed for improving both the accuracy and scalability of unsupervised graph embedding algorithms, which is a multi-level spectral framework. In addition to designing specific algorithms and models, there are also a few third-party libraries like DGL [49] for users to make the network scalable.

## 2.3 Layout Decomposition and Mask Optimization

Due to the delay of the next generation of lithography techniques, the current lithography wavelength is stuck at 193nm. As a result, resolution enhancement techniques

(RETs) on layout and mask are of great importance to improve the yield. Multiple patterning lithography (MPL) has achieved great success in pushing forward the technology node. There are two of the most critical stages in the MPL process, including layout decomposition and mask optimization. In layout decomposition, the target layout is decomposed into several layouts so that each decomposed layout can be manufactured under the current lithography condition. Two main types of MPL manufacturing processes are litho-etch-litho-etch (LELE)-type MPL and spacer-type MPL. Spacer-type MPL typically refers to self-aligned double patterning (SADP). LELE-type refers to conventional double patterning layout decomposition (DPLD) or triple patterning layout decomposition (TPLD), depending on the number of masks available.

To achieve high efficiency and maintain high solution quality, a variety of decomposition algorithms have been proposed. These algorithms can be roughly categorized into three types, including mathematical programming and relaxation, graph-theoretical approaches, and search-based approaches. Mathematical programming solves the MPLD problem by formulating it into a standard optimization model. Integer linear programming (ILP) is adopted to solve the problem of layout decomposition, including DPLD [50, 51, 52] and TPLD [53, 54].

In order to deal with the runtime overhead of solving an ILP, relaxation is a commonly used approach. It can provide an upper bound or a lower bound on the optimal value of the original problem. A representative relaxation is semidefinite programming (SDP) relaxation [53], which can be solved in polynomial time. Lin *et al.* [55] proposed to apply linear programming (LP) relaxation to avoid the infeasibility issue and find a solution with few conflicts. Li *et al.* [56] proposed a discrete relaxation method for TPLD problem. Firstly, the original TPLD problem is relaxed to an ILP by ignoring stitch insertion, whose optimal value can be treated as a lower bound of the optimal value of the original TPLD problem. The ILP formulation of the relaxed problem has fewer variables and fewer constraints than the reduced version of [53]. Another category is to directly perform color assignment based on a set of graph-theoretical algorithms, e.g., the maximal independent set (MIS) [57], the shortest-path [58, 59], and fixed-parameter tractable (FPT) algorithms [60]. Search-based algorithms follow a divide-and-conquer principle with each sub-graph containing a small number of nodes, e.g., less than 20. Then a search procedure is applied to find the optimal solutions for small sub-graphs [61, 57, 62, 53, 63, 64].

The diffraction effect of the light cannot be ignored since the size of the patterns on a layout is comparable with the wavelength of the lithography light source, which may result in low fidelity of the final on-wafer image. In mask optimization, e.g., optical proximity correction (OPC), each mask is refined to compensate for the diffraction effect of the light in advance to ensure the high quality of the on-wafer image. Finally, all optimized masks go through the lithography process separately, then all printed images are combined together to generate the target image.

Nowadays, there are three sorts of OPC methodologies, including model-based OPC, inverse lithography technique, and learning-based OPC. A model-based flow is presented by Awad *et al.* [65] for maximizing the printability. Kuang *et al.* [66] propose an acceleration trick to tackle the long runtime of simulation under multiple process corners. Su *et al.* [67] develop a model-based OPC flow, PVOPC, in which a novel dynamic edge fragmentation is used to form segment candidates for correction. For ILT, it aims to find the ideal mask by solving an inverse problem of the lithography system. The objective of ILT is typically to minimize the difference between printed patterns of the mask and target patterns. Different from model-based OPC which makes correction on edge segments, ILT is based on pixel-based representation. Poonawala *et al.* [68] adopt relax the problem to continuous form and solve it by gradient descent. In addition to conventional gradient descent algorithm, stochastic gradient descent (SGD) is also used in ILT [69]. Another acceleration technique for ILT is proposed [5]. Considering that the forward lithography model can be formulated by a series of weighted sum of convolution, the "effective kernel" can be precomputed without loss of accuracy. Shen *et al.* [70] shows that ILT can be modeled into an image restoration problem and can be solved by the level-set method. Both the first-order accurate method [70] and the conjugate gradient method [71] are applied in the level-set method. For learning-based OPC approaches, there are several attempts exploring to perform pixel-wise or segment-wise correction on the mask [72, 73, 74]. GAN-OPC [75] firstly introduces the generative adversarial network for mask optimization, which takes target circuit patterns as input and generates quasi-optimal masks for post-refinement.

## 2.4 Hardware-Friendly Learning

As neural networks become deeper and deeper, the representation ability of neural network keeps improving, leading to significant performance promotion in a variety of tasks. However, the model size and the computation cost of neural networks are also increasing due to the huge amount of weights learned, which results in low throughput in the inference stage and restrains the deployment on resource-limited systems. For example, embedded devices may lack enough storage and computation power to execute the giant networks. Meanwhile, deep neural networks are demonstrated to be over-parameterized [76], which motivates researchers to explore efficient approaches to make the deep models compact.

Low-rankness and sparse connections are the most commonly applied assumptions when approximating a model. The sparse connection can be realized by pruning a pre-trained network, which is the most straightforward approach. The majority of the parameters in a sparse layer are zeros, thus the parameters can be stored with compressed representation, e.g., compressed sparse row (CSR) format, for size reduction. A hard thresholding approach is proposed in [77], which achieves high sparsity by removing the weights with less importance. However, the sparse pattern is non-structured which has limited benefits for speedup during inference due to the poor weight locality. [78] proposes to prune the entire convolution kernel rather than a single element based on the intensity. A structured sparse learning algorithm is proposed in [79], which enables us to learn a network with a structured sparse network by applying group sparse regularizations during training. Since structured sparsity leads to zero-columns and zero-rows in the lowered matrices, [79] further proposes to reduce the dimension of lowered matrices by removing these zero-columns and zero-rows, which reduces the dimension of the lowered weight matrix when applying General Matrix-Matrix Multiplication (GEMM) function and accelerates inference. A channel pruning method is proposed in [80], which can be considered as a special case of structured sparsity. The difference is that channel pruning is performed on a pre-trained model rather than training the model from scratch. [80] formulated the problem as $l_0$-norm minimization problem, trying to find the "informative" channels of the feature map and the corresponding weights. Instead of trying to minimize the reconstruction error layer by layer, [81] targets at a unified goal which is to minimize the reconstruction error of important response in the final response layer.

In addition to sparsifying a network, the low-rank approximation is another sort of approach which can be applied for both network compression and acceleration. In modern convolutional neural networks (CNNs) structure, filters are usually a 4-D tensor. Some tensor decomposition techniques are leveraged for acceleration and compression. A straightforward idea is to replace the 4-D tensor with two consecutive tensors with lower-rank [82]. In addition, other kinds of tensor decomposition can also be applied. In [83], fully-connected layers are converted to the Tensor Train format, resulting in compression by a huge factor. CP-decomposition of the filter tensors is proposed in [84]. A relevant approach to low-rank approximation is tensor sketching [85]. The difference is that low-rank approximation will increase the network depth since an original layer will be decomposed into multiple layers. In order to conduct low-rank approximation more efficiently, methods for training neural networks with low-rank filters are investigated [86, 87, 88, 89].

Domain adaptation is a solution to reduce the need and effort to collect the training data [90]. Since the main issue is the distribution discrepancy across different domains, many methods are proposed to match the feature distributions in the source and the target domains [91, 92]. Recently, more efforts on unsupervised domain adaptation with deep learning methods have been witnessed. The first category is based on explicit distribution matching with a well-defined criterion, e.g., MMD [93, 94, 95] and central moment discrepancy (CMD) [96]. Alternatively, the adversarial training scheme is investigated by leveraging a domain discriminator [97, 98, 99], assuming that a good representation for domain transfer is one that an algorithm cannot distinguish the origin domain of the input observation. These methods are based on an assumption that the label space is fully shared between the source domain and the target domain, which may not always hold in PDA. The adversarial training scheme has been studied for tackling the PDA problem [100, 101, 102, 103], which achieves the state-of-the-art performance by introducing dedicated re-weighting mechanisms over the instances or classes. Regarding the hardware-friendly PDA applications, previous works on network compression focus more on the conventional supervised learning tasks in a single domain, and there are bare of studies showing how network compression can help in unsupervised domain adaptation tasks. A recent work [104] proposed a transfer channel pruning approach for domain adaptation models by removing less important channels iteratively. However, only identical label space setting is explored in [104].

# Chapter 3

# Active Learning for Design Space Exploration

In the last decades, the industrial EDA tools have advanced towards optimality, especially at the individual stages of the VLSI design cycle. Nevertheless, with growing design complexity and aggressive technology scaling, physical design issues have become more and more complex. As a result, the constraints and the objectives of higher layers, such as the system or logic level, are very difficult to be mapped into those of lower layers, such as physical design, and vice-versa, thereby creating a *gap* between the optimality at the logic stage and the physical design stage. This necessitates the innovation of data-driven methodologies, such as machine learning [105, 106, 10, 107, 108], to bridge this gap.

Adder design is one of the fundamental problems in the digital semiconductor industry, and its main bottleneck (in terms of both delay and area) is the carry-propagation unit. This unit can be realized by hundreds of thousands of parallel prefix structures, but it is hard to evaluate the final metrics without running through physical design

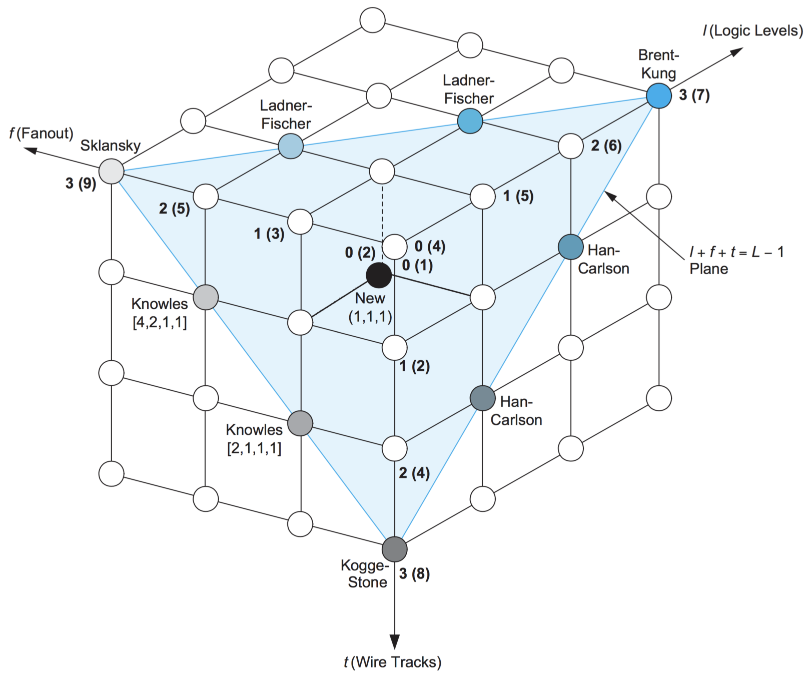


Figure 3.1: Regular adders (picture taken from [3]).

tools. Historically, regular adders [109, 110, 111, 112] have been proposed for achieving the corner points in terms of various metrics as shown in Figure 3.1 in the architectural stage.

To address this gap between prefix adder synthesis and actual physical design of the adders, one approach is to explore a huge design space efficiently. An exhaustive bottom-up enumeration technique with several pruning strategies are investigated to generate innumerable prefix structure solutions [1]. However, it is very hard to analytically model the physical design complexities, such as wire-length and congestion issues, the physical design metrics, such as the area, power, delay, etc., may not be mapped well to the prefix structure metrics, such as the size, max-fan-out ($mfo$), etc. Although the performance can be evaluated with commercial synthesis tools, it is computationally very intensive to run all solutions through synthesis, placement, and routing. To tackle the high computational effort during the synthesis flow, machine learning methodologies are leveraged to perform the design space exploration. Our main contributions are summarized as follows:

- A comprehensive framework for optimal adder search by machine learning methodology bridging the prefix architecture synthesis to the final physical design;
- A machine learning model for prefix adders, guided by quasi-random data sampling with features considering architectural attributes and EDA tool settings;
- A design space exploration method to generate the Pareto frontier for delay vs. power/area over a wide design space;
- An active learning approach for the design space exploration, which uses less labeled data and achieves better quality of Pareto frontier.

The rest of the chapter is organized as follows. Section 3.1 presents the background of prefix adder synthesis, Next, two machine learning approaches of design space exploration for high-performance adders are described. Section 3.3 presents the passive supervised learning, while Section 3.4 introduces a Pareto frontier-driven active learning approach. Section 3.5 lists the experimental results, followed by a summary in Section 3.6.

## 3.1 Preliminaries

### 3.1.1 Prefix Adder Synthesis

An $n$ bit adder accepts two $n$ bit addends $A = a_{n-1}..a_1a_0$ and $B = b_{n-1}..b_1b_0$ as input, and computes the output sum $S = s_{n-1}..s_1s_0$ and carry out $C_{out} = c_{n-1}$, where

$s_i = a_i \oplus b_i \oplus c_{i-1}$ and $c_i = a_i b_i + a_i c_{i-1} + b_i c_{i-1}$. The simplest realization for the adder network is the ripple-carry-adder but with logic level $n-1$, which is too slow. For faster implementation, the carry-lookahead principle is used to compute the carry bits. Mathematically, this can be represented with bitwise (group) generate function $g$ ($G$) and propagate function $p$ ($P$) by the Weinberger's recurrence equations as follows [113]:

- Pre-processing (inputs): Bitwise generation of $g$, $p$

$$g_i = a_i \cdot b_i \text{ and } p_i = a_i \oplus b_i. \tag{3.1}$$

- Prefix processing: This part is the main carry-propagation component where the concept of generate/propagate is extended to multiple bits and $G_{[i:j]}$, $P_{[i:j]}$ ($i \geq j$) are defined as

$$P_{[i:j]} = \begin{cases} p_i, & \text{if } i = j, \\ P_{[i:k]} \cdot P_{[k-1:j]}, & \text{otherwise}, \end{cases} \tag{3.2}$$

$$G_{[i:j]} = \begin{cases} g_i, & \text{if } i = j, \\ G_{[i:k]} + P_{[i:k]} \cdot G_{[k-1:j]}, & \text{otherwise}. \end{cases} \tag{3.3}$$

The associative operation $\circ$ is defined for ($G$, $P$) as:

$$\begin{aligned} (G,P)_{[i:j]} &= (G,P)_{[i:k]} \circ (G,P)_{[k-1:j]} \\ &= (G_{[i:k]} + P_{[i:k]} \cdot G_{[k-1:j]}, P_{[i:k]} \cdot P_{[k-1:j]}). \end{aligned} \tag{3.4}$$

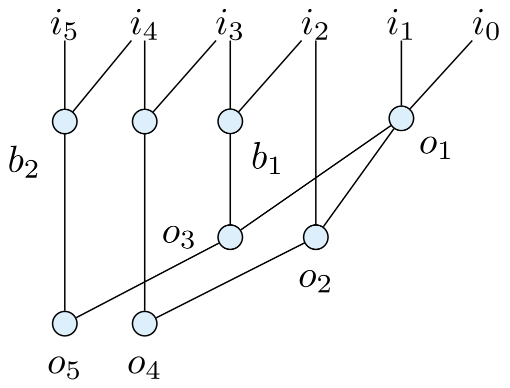


Figure 3.2: 6 bit prefix adder network.

- Post-processing (outputs): Sum/Carry-out generation

$$s_i = p_i \oplus c_{i-1}, \quad c_i = G_{[i:0]}, \text{ and } C_{out} = c_{n-1}. \tag{3.5}$$

The ‘Prefix processing’ or carry propagation network can be mapped to a prefix graph problem with inputs $i_k = (p_k, g_k)$ and outputs $o_k = c_k$, such that $o_k$ depends on all previous inputs $i_j$ $(j \le k)$. Any node except the input nodes is called a *prefix* node. Size of the prefix graph is defined as the number of prefix nodes in the graph. Figure 3.2 shows an example of such prefix graph of 6 bit and we can see that $C_{out} = c_5 = o_5$ is given by

$$o_5 = (i_5 \circ i_4) \circ ((i_3 \circ i_2) \circ (i_1 \circ i_0)). \tag{3.6}$$

Size ($s$), logic level ($L$) and maximum-fan-out ($mfo$) for this network are respectively 8, 3 and 2. Note that here the number of fan-ins for each of the associative operation $o$ is two, thus this is called radix-2 implementation of the prefix graph. However, there exist other options such as radix-3 or radix-4, but the complexity is very high and not beneficial in static CMOS circuits [114]. In this work, the logic levels for all output bits are $\log_2 n$, *i.e.*, the minimum possible, to target high-performance adders.

### 3.1.2 Pareto Optimality

**Definition 1 (Pareto Optimality)** *An objective vector* $\mathbf{f}(\mathbf{x})$ *is said to dominate* $\mathbf{f}(\mathbf{x}')$ *if:*

$$\begin{aligned} &\forall i \in [1,n], f_i(\mathbf{x}) \leq f_i(\mathbf{x}') \\ &\textit{and } \exists j \in [1,n], f_j(\mathbf{x}) < f_j(\mathbf{x}'). \end{aligned} \tag{3.7}$$

A point x is *Pareto-optimal* if there is no other $\mathbf{x}'$ in design space such that $\mathbf{f}(\mathbf{x}')$ *dominates* $\mathbf{f}(\mathbf{x})$.

**Definition 2 (Hypervolume)** *The hypervolume computes the volume enclosed by the Pareto frontier and the reference point in the objective space [115].*

In Figure 3.3, the shaded area is an example of the hypervolume of a Pareto set with two objectives. Then the *hypervolume error* for a predicted Pareto set $\hat{P}$ is defined as

$$\eta = \frac{V(P) - V(\hat{P})}{V(P)}, \tag{3.8}$$

where $P$ is the true Pareto-optimal set, and $V(P)$ is the hypervolume of the Pareto set $P$. Note that a prediction $\hat{P}$ which contains the whole design space has an error of 0. Thus the predicted set $\hat{P}$ with fewer points is desired.

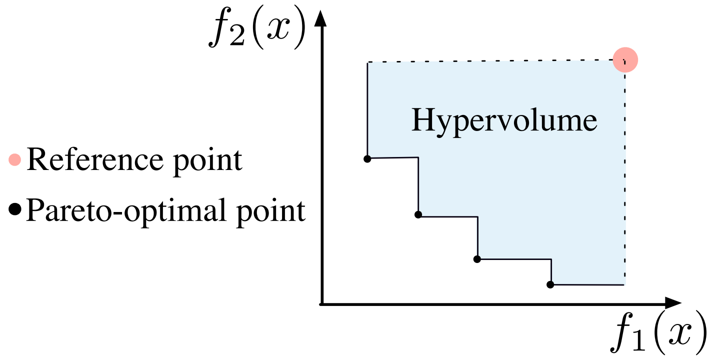


Figure 3.3: Hypervolume with two objectives in objective space.

## 3.2 Bridging Architectural Solution Space to Physical Solution Space

In most EDA problems, the metrics of the solution quality are typically conflicting. For instance, if we optimize the timing of the design, then the power/area may be compromised and vice versa. So one imperative job of EDA engineers is to find the Pareto-optimal points of the design enabling the designers to select among those. In this section, we first provide the preliminaries about Pareto optimality and the error metrics of Pareto optimal solutions. Then we discuss the gap between the prefix architectural solution space and physical solution space in adders, which motivates the need for the machine learning-based approach for optimal adder exploration. Finally, a domain knowledge-based feature selection details are presented along with training data sampling for the learning models.

As in this work for adder design, a Pareto-optimal design is where none of the objective metrics, such as area, power, or delay, can be improved without worsening at least one of the others. The *Pareto Frontier* is the set of all the Pareto-optimal designs in the *objective space*. Therefore, the goal is to identify the Pareto-optimal set $P$ for

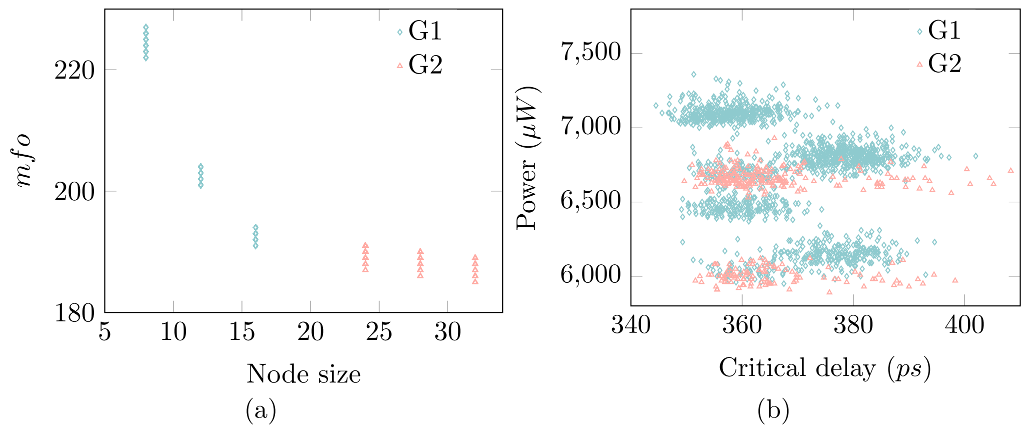


Figure 3.4: Gap between prefix structure and physical design of adders: (a) Architectural solution space; (b) Physical solution space.

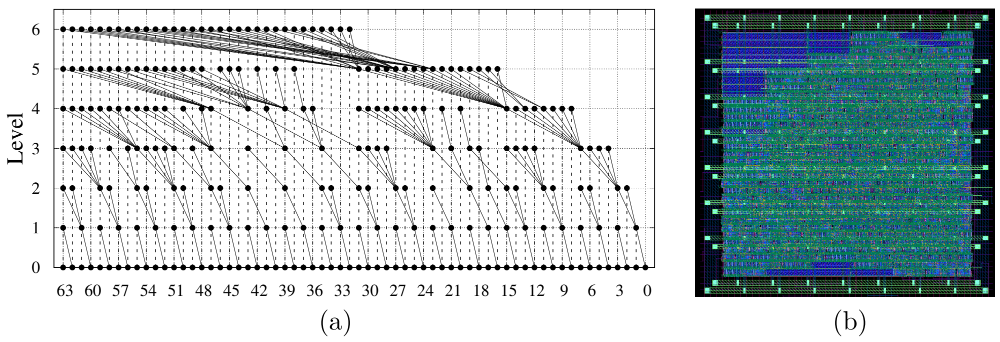


Figure 3.5: (a) An example of architectural solution: Bit-width = 64, size = 201, Max. level= 6, Max. fanout = 12; (b) Corresponding physical solution.

all the Pareto-optimal designs.

### 3.2.1 Gap Between Logic and Physical Design

Since we focus on high-performance adders and explore the prefix adders of logic level $L = \log_2 n$, the metrics at this architecture stage are prefix node size $s$ and max fanout $mfo$. These two metrics are conflicting, *i.e.*, if we reduce $mfo$, $s$ increases and vice-versa. A similar competing relationship exists between delay and power/area af-

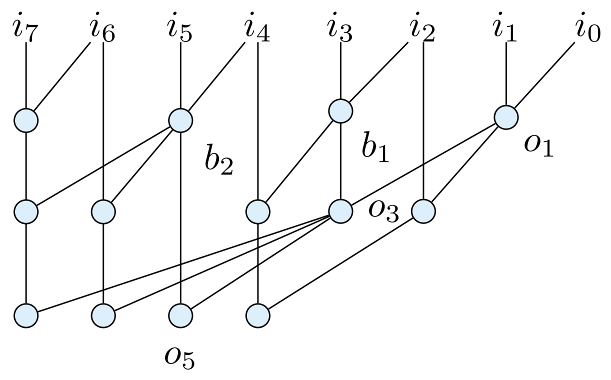


Figure 3.6: Defining $spfo$ of a node.

ter physical design. It should be stressed that power and $s$ are correlated, and $mfo$ indirectly controls the timing as a more restricted fan-out can mitigate congestion and load-distribution, thereby improving the delay of the adder. However, this relationship between architectural synthesis and physical design is approximate, and not a very high-fidelity one.

To demonstrate this, we plot node size $s$ vs. $mfo$ and power vs. delay in Figure 3.4 for several 64 bit adder solutions. In this experiment, we generated the prefix architecture solutions with prefix graph generation algorithms, and the final power/delay numbers are obtained by running those solutions through EDA tools as explained later in Section 3.5. An example of the prefix architecture and the corresponding physical solution is presented in Figure 3.5. In Figure 3.4a, we broadly categorize the solutions into 2 groups, (i) $G_1$ with higher node size and lower $mfo$, and (ii) $G_2$ with lower node size and higher $mfo$. In Figure 3.4b, the same designs as Figure 3.4a are projected into the physical solution space, restoring the group information. Design Compiler [116] (version F-2011.09-SP3) is used for logical synthesis, and IC Compiler [117] (version J-2014.09-SP5-3) is used for the placement and routing. Non-linear delay model (NLDM) in 32$nm$ SAED cell-library [118] is used for technology mapping. The key observations here are firstly, *there is a correlation between architectural solution space and physical design solution space.* For instance, the solutions from $G_1$ are mostly on the upper side, and those of $G_2$ are mostly on the lower side in Figure 3.4b, thereby indicating a correspondence between $s$ and power. Nevertheless, *it is not completely reliable.* For example, (i) the delay numbers for $G_1$ and $G_2$ are very much spread, (ii) a cluster can be observed where the solutions from $G_1$ and $G_2$ are mixed up in Figure 3.4b, and (iii) several solutions of $G_1$ are better than several solutions of $G_2$ in power, which is not in accordance with the metrics at the prefix adder architecture stage. So we can not utterly rely on architectural solution space to achieve the optimal output in physical solution space.

However, since our algorithm generates **hundreds of thousands** of prefix graph structures, it is intractable to run the synthesis and physical design flows for even a small percentage of all available prefix adder architectures. To address this **fidelity gap** between the two design stages and the high computational cost together, we come up with a novel machine learning guided design space exploration as a replacement of exhaustive search.

### 3.2.2 Feature Selection

The feature is a representation which is extracted from the original input representation, and it plays an important role in machine learning tasks. We now discuss the features to be used for the learning model. Features are considered from both prefix adder structure and tool settings, with a focus on the former. We select node size and maximum-fan-out ($mfo$) of a prefix adder as two main features for our learning model. However, for any given $mfo$ and node size, there will be hundreds or even thousands of different prefix architectures. Therefore, additional features are required to better distinguish individual prefix adder attributes. We define a parameter sum-path-fan-out ($spfo$) for this. Let $a$ and $b$ are the fan-in nodes of a node $n$, then $spfo(n)$ is defined recursively as:

$$spfo(n) = \begin{cases} 0, & \text{if } n \in \text{input}, \\ \text{sum}(fo(a) + spfo(a), & \\ \qquad fo(b) + spfo(b)), & \text{otherwise.} \end{cases} \tag{3.9}$$

Here $fo(n)$ denotes the fan-out of any node $n$. Consider the prefix adder structure in Figure 3.6, and according to the definition we have:

$$\begin{aligned} spfo(o_1) &= \text{sum}(fo(i_0) + spfo(i_0), fo(i_1) + spfo(i_1)) \\ &= \text{sum}(1, 1) = 2, \\ spfo(b_1) &= \text{sum}(fo(i_2) + spfo(i_2), fo(i_3) + spfo(i_3)) \\ &= \text{sum}(2, 1) = 3, \\ spfo(b_2) &= \text{sum}(fo(i_4) + spfo(i_4), fo(i_5) + spfo(i_5)) \\ &= \text{sum}(2, 1) = 3. \end{aligned}$$

Therefore, we can use the recursive definition to calculate

$$\begin{aligned} spfo(o_3) &= \text{sum}(fo(o_1) + spfo(o_1), fo(b_1) + spfo(b_1)) \\ &= \text{sum}(3 + 2, 2 + 3) = 10, \\ spfo(o_5) &= \text{sum}(fo(o_3) + spfo(o_3), fo(b_2) + spfo(b_2)) \\ &= \text{sum}(3 + 10, 3 + 3) = 19. \end{aligned}$$

In our methodology, we use the $spfo$ of the output nodes which are at $\log_2 n$ level (there are 32 nodes at level 6 for 64 bit adder) as the features to characterize the prefix structures, in addition to $mfo$, size and target delay. The basic intuition for selecting $spfo$ of the output nodes as the features is that the critical path delay of the adder is the longest path delay from input to output. So it depends on the (i) path-lengths, which can be represented at the prefix graph stage by the logic level of the node, and (ii) the number of fan-outs driven at every node on the path. Note that we have skipped the $spfo$ of the output nodes which are not at $\log_2 n$ level as for those nodes, the path length is smaller, and those would not potentially dictate the critical path delay.

Apart from these prefix graph structural features, we also consider tool settings from the synthesis stage and physical design stage as other features. We have synthesized the adder structures using industry-standard EDA synthesis tool [116], where we can specify the target-delay for the adder. The tool then adopts different strategies internally to meet that target-delay which we can hardly take into account during prefix graph synthesis. Consequently, changing target-delay can lead to different power/timing/area metrics. So we have considered target-delay as a feature in our learning approach.

In physical design, utilization is an important parameter, which defines the area occupied by standard cell, macros, and blockages. Different utilization values can lead to different layouts after physical design. Therefore, we take utilization as another feature in the learning model.

In addition to the target delay and utilization, other tool settings have also been explored. The optimization level setting in logical synthesis has a potential impact on the performance of adders, which can be adjusted by `compile` and `compile_ultra` commands with different options. After synthesizing, it is observed that the solutions generated with `compile_ultra` can significantly dominate the solutions generated by `compile`. Therefore, this setting is fixed to `compile_ultra` level as we are aiming at superior designs.

In this work, the technology node is not used as a feature. From the machine learning perspective, there is a common assumption for conventional machine learning applications that the training and test data are drawn from the same feature space and the same distribution [90]. The values of area/power/delay may vary a lot under different technology nodes, which results in different underlying data distributions. Therefore, the technology node for synthesis should be consistent. The proposed approach for feature extraction can also be applied to other technology nodes as long as the technology node is consistent during the design flow. If the technology node of the testing data switches to another one, the machine learning model should be re-trained using the data from that technology node to ensure the accuracy of the model.

### 3.2.3 Data Sampling

Since we can not afford to run the physical design flow for too many architectures, and too few training data may degrade the model accuracy significantly, a set of adders need to be selected to represent the entire design solution space. However, finding a succinct set of representative training data for traditional supervised learning is difficult. In order to tackle this difficulty, we come up with two learning approaches in the next two sections. The first one is the passive supervised learning where a quasi-random data sampling is performed to obtain the training data, followed by multi-objective scalarization to achieve Pareto optimal solutions. The second one is the active learning approach where model training is integrated into finding Pareto-optimal frontiers of the design space.

## 3.3 $\alpha$-sweep learning

In this section we propose a pareto-frontier exploration flow which is based on support vector machine. The overall flow of our $\alpha$-sweep supervised learning-based Pareto-frontier exploration is presented in Figure 3.7.

### 3.3.1 Scalarization to the Single-Objective

In this work, supervised learning is preferred over unsupervised learning since supervised learning has a substantial advantage over unsupervised learning for our problem. In particular, supervised learning allows to take advantage of the golden result, i.e., the true area/power/delay, generated by the synthesis tools for each design, instead of just letting the algorithm work out for itself what the classes should be. In general, supervised learning usually outperforms the unsupervised learning for this kind of regression and classification tasks.

Before applying machine learning for exploring the Pareto frontier, we first validate the effectiveness of the features we extract by building regression models for single metric prediction. For learning models, we explored (i) several supervised learning techniques, such as linear regression, Lasso/Ridge, Bayesian ridge model and support vector regression (SVR) with linear, polynomial and radial-basis-function (RBF) kernel, and (ii) 36 features, including 4 primary features, size, $mfo$, target delay and utilization (tool settings), and 32 secondary features for $spfo$. We observed that we could get an $R^2$ score above 0.95 for area and power even with primary features and linear models.

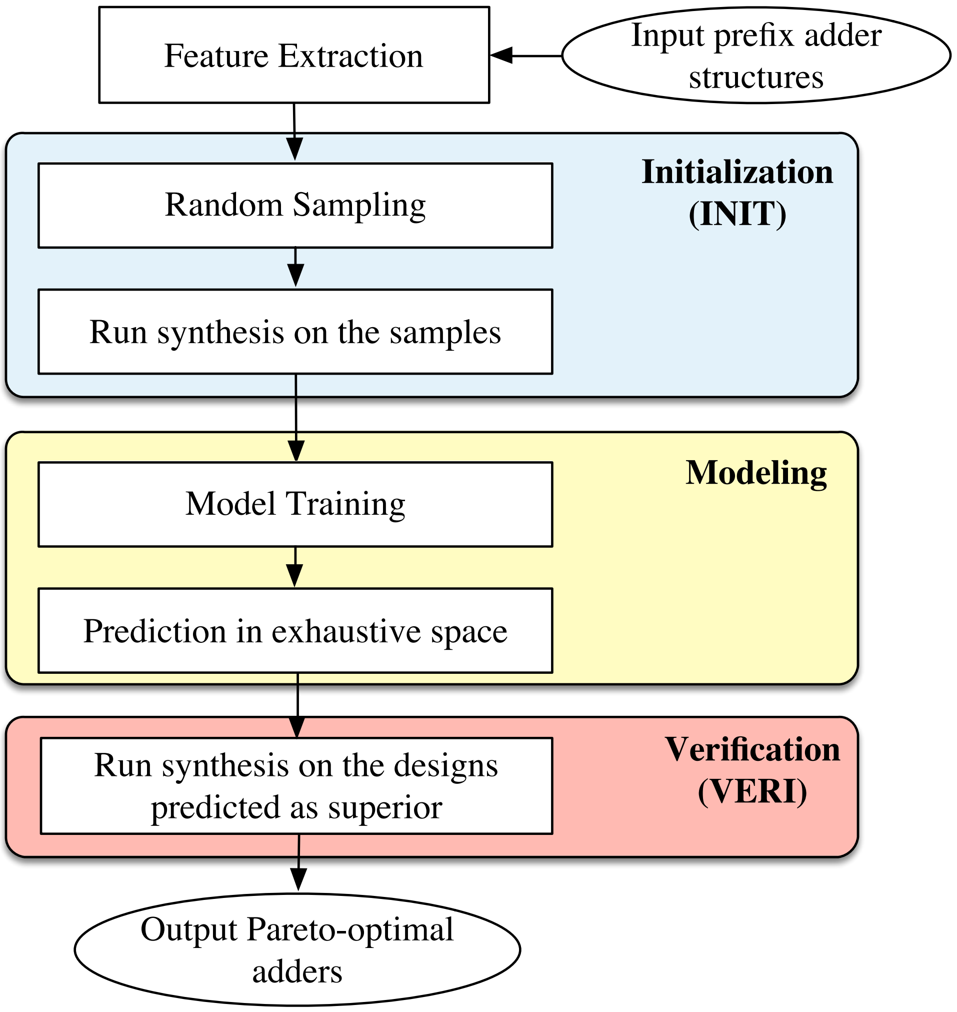

Feature Extraction
Input prefix adder structures
Initialization (INIT)
Random Sampling
Run synthesis on the samples
Modeling
Model Training
Prediction in exhaustive space
Verification (VERI)
Run synthesis on the designs predicted as superior
Output Pareto-optimal adders

Figure 3.7: Overall flow of $\alpha$-sweep learning.

However, we don't get good scores for delays with only primary features. The best model fitting for the delay is achieved with SVR (RBF kernel) with these 4 primary and 32 secondary features. Since SVR with RBF kernel gives good MSE (mean-squared-error) scores for all metrics, delay, area, and power, we have used this model throughout for design space exploration.

The model experiments give us the following key insights: (i) **tool setting** can play an important role in building the learning models in EDA. For instance, MSE scores for area and power improve from 0.021 to 0.003, and 0.228 to 0.027 respectively when we add the 'target delay' feature in our model building, (ii) secondary features play an important role in improving model accuracy. For instance, when we include $spfo$ features in model building, the MSE score for delay improves from 0.200 to 0.170. (iii) linear models are not sufficient for modeling delay. For instance, MSE scores of delay improve from 0.214 to 0.170 when we go from linear models to SVR with RBF kernel, with the same set of features.

The problem of exploring the Pareto frontier of rich prefix adder space can be approached by first sampling a subset of prefix adder architectures, and generating the power, area, delay numbers of each prefix adder by running through the logic synthesis and physical design flow. Those known data set will be used as the training and testing data for supervised machine learning guided model fitting. Once the model is fitted, we can apply the exhaustive prefix adder architectures to this model and get the predicted Pareto frontier solution set. This is due to the merit of much faster runtime for a machine learning model in the prediction stage than running the entire VLSI CAD flow.

However, the conventional machine learning problem aims at maximizing the prediction accuracy rather than exploring a Pareto frontier out of a solution set. Improving the model accuracy does not necessarily improve the Pareto frontier and the direct use of the fitted model for Pareto frontier exploration can even miss up to 60% Pareto frontier points [10]. We therefore need a machine learning integrated Pareto frontier exploration methodology, where the Pareto frontier selection does not rely only on the model accuracy. So we develop a fast yet effective algorithmic methodology, enabled by the regression model to explore the Pareto frontier of prefix adder solutions.

First, we consider two spaces for Pareto frontier exploration: the delay vs. area as well as the delay vs. power. For either space, there exists a strong trade-off between the two metrics. For delay vs. power space, we propose to use a joint output Power-Delay function ($PD$) as the regression output rather than using any single output.

$$PD = \alpha \cdot Power + Delay. \tag{3.10}$$

The rationale of using scalarization [119] or the linear summation of the power and delay metrics is that such a linear relation provides a weighted bonding between the power and the delay so that by changing the $\alpha$ value, the regression model will try to minimize the prediction error on the more weighted axis hence leads to more accuracy in that direction. In contrast, the other metric direction will be predicted with less accuracy hence introducing some level of relaxations. It can be foreseen that changing the $\alpha$ value can lead to different fitting accuracies of the regression model. By sweeping $\alpha$ over a wide range from 0 to large positive values, each time the regression model will be fitted to predict different best solutions which altogether form the Pareto frontier. We call this approach **$\boldsymbol{\alpha}$-sweep**. Note that, the $Power$ and $Delay$ values in Equation (3.10) are normalized and scaled to the range between 0 and 1 by Equation (3.11).

$$x = \frac{x - \min(X)}{\max(X) - \min(X)}, x \in X. \tag{3.11}$$

Similarly, we have a joint output Area-Delay (AD) function for Pareto frontier exploration on Area and Delay space.

$$AD = \alpha \cdot Area + Delay. \tag{3.12}$$

This $\alpha$-sweep technique can be extended to simultaneously consider power, performance or delay, and area (PPA), using two scalars ($\alpha_1$ and $\alpha_2$) instead of one scalar factor $\alpha$. The joint output function for Pareto frontier exploration on the area – power – delay space can be formulated as:

$$PPA = \alpha_1 \cdot Area + \alpha_2 \cdot Power + Delay. \tag{3.13}$$

The results of $\alpha$-sweep for both two-dimensional space and three-dimensional space are shown in the Section 3.5.

## 3.4 Pareto Active Learning

In our adder design problem, obtaining the true area/power/delay values or the labeled data for each adder requires running logic synthesis and physical design flow, which is often time-consuming if the amount of data is huge. Active learning is iterative supervised learning which is able to interactively query the data pool to obtain the desired outputs at new data points. Since the samples are selected by the learning algorithm, the number of samples to fit a model can often be much lower than the number required in traditional supervised learning. Since an active sampling strategy is required in active learning, an “uncertainty estimation” of the prediction is needed. Gaussian Process (GP) can make predictions and, more importantly, provide the uncertainty estimation of its predictions by nature. Therefore, in this work, we further propose a Pareto active learning algorithm based on Gaussian Process regression.

### 3.4.1 Overall Flow

The overall flow of the Pareto active learning (PAL) is shown in Figure 3.8. Given all the prefix adder structures, first, we extract the feature vector for each adder as introduced in Section 3.2.2. The active learning starts with Gaussian Process regression which will be illustrated later. Unlike the passive supervised learning in which all the features and the corresponding labels are prepared in advance, the active learning derives the labels of each training data during the learning process on-demand. To be specific, the algorithm incrementally identifies the most representative instances along with their features which are later fed into EDA synthesis flow (synthesis, placement, and routing) for true area/power/delay numbers. Namely, the EDA synthesis flow and the learning process are interleaving. As more and more designs being selected, the model gets more and more accurate till convergence.

### 3.4.2 Gaussian Process Prediction

A Gaussian process is specified by its *mean function* and *covariance function.* A Pareto active learning scheme based on Gaussian process regression is proposed in [27]. The prior information is important to train the Gaussian Process model, which is a parame-

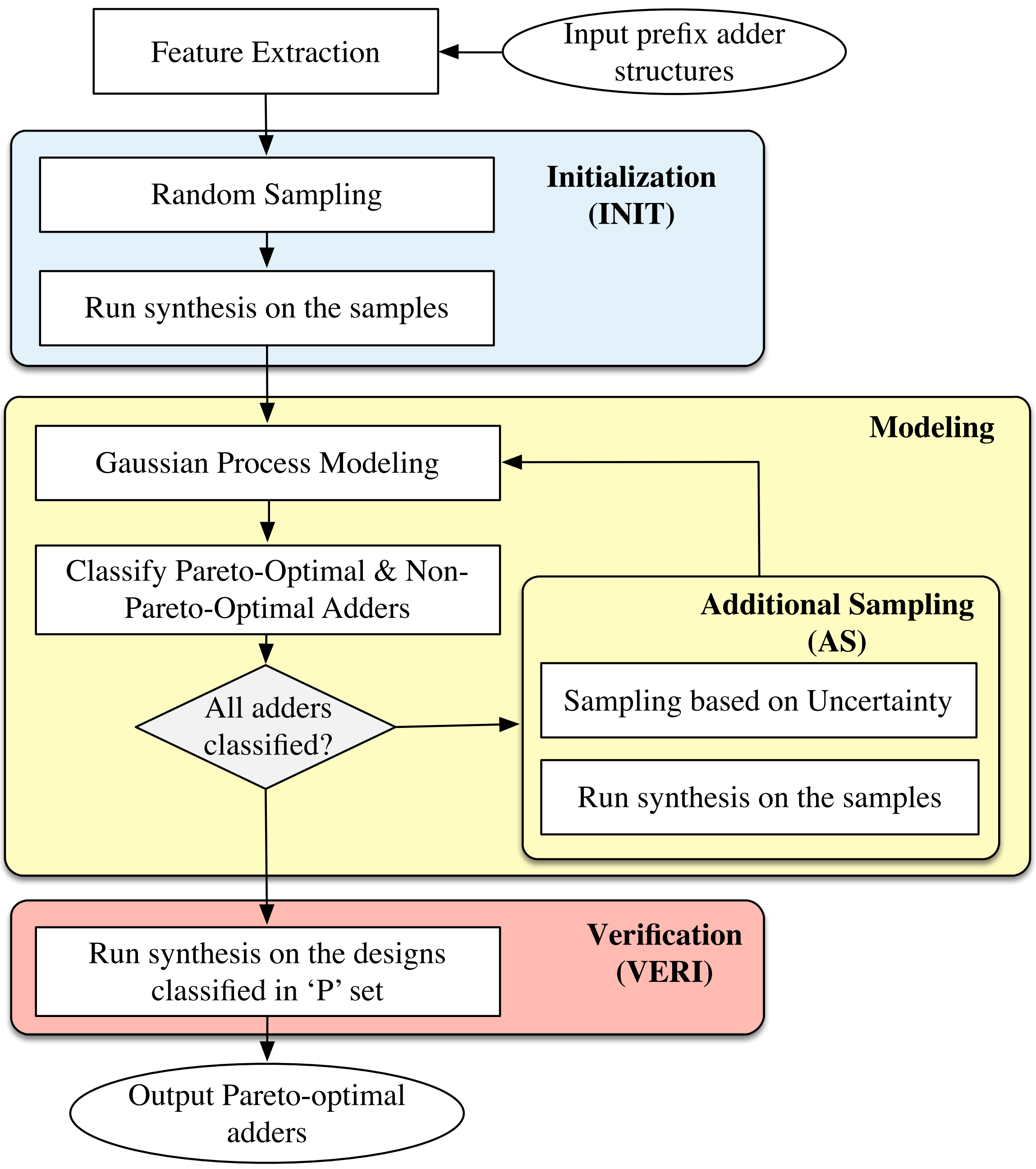

Feature Extraction
Input prefix adder structures
Initialization (INIT)
Random Sampling
Run synthesis on the samples
Modeling
Gaussian Process Modeling
Classify Pareto-Optimal & Non-Pareto-Optimal Adders
All adders classified?
Additional Sampling (AS)
Sampling based on Uncertainty
Run synthesis on the samples
Verification (VERI)
Run synthesis on the designs classified in 'P' set
Output Pareto-optimal adders

Figure 3.8: Overall flow of Pareto active learning.

terized mean and covariance functions. Conventionally, the training process selects the parameters in the light of training data such that the marginal likelihood is maximized.

Then the Gaussian Process model can be obtained and the regression can proceed with supervised input [18]. The ability of GP indicating prediction uncertainty reflects in GP learner providing a Gaussian distribution $\mathcal{N}(m(\mathbf{x}), \sigma(\mathbf{x}))$ of the values predicted for any test input $\mathbf{x}$ by computing

$$\begin{aligned} m(\mathbf{x}) &= k(\mathbf{x}, \mathbf{X})^\top (k(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I})^{-1} \mathbf{Y}, \\ \sigma^2(\mathbf{x}) &= k(\mathbf{x}, \mathbf{x}) - k(\mathbf{x}, \mathbf{X})^\top (k(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I})^{-1} k(\mathbf{x}, \mathbf{X}), \end{aligned} \tag{3.14}$$

where $\mathbf{X}$ is the training set, $\mathbf{Y}$ is the supervised information of trained set $\mathbf{X}$. For Gaussian Process regression, a prediction of a design objective consists of a mean and a variance. The mean value $m(\mathbf{x})$ represents the predicted value and the variance $\sigma(\mathbf{x})$ represents the uncertainty of the prediction.

### 3.4.3 Active Learning Algorithm

The ability of GP learners in quantifying prediction uncertainty enables a suitable application for active learning. Basically, three sets are maintained during the PAL process, including a set of Pareto-optimal designs ($P$), non-Pareto-optimal designs ($N$) and ‘unclassified’ designs ($U$).

The GP models with discrepant prior are applied to learn the objective functions $f_{area}(\mathbf{x})$, $f_{power}(\mathbf{x})$, $f_{delay}(\mathbf{x})$. PAL calls GP inference to predict the mean vector $\mathbf{m}(\mathbf{x})$ and the standard deviation vector $\boldsymbol{\sigma}(\mathbf{x})$ of all unsampled $\mathbf{x}$ in the design space based on Equation (3.14). Unlike other regression models such as linear regression and support vector regression, whose outputs are in form of numerical or categorical results, the output of GP is a distribution where uncertainties are involved. To capture the prediction uncertainty for a design $\mathbf{x}$, a hyper-rectangle is defined as

$$HR(\mathbf{x}) = \{\mathbf{y} : m_i(\mathbf{x}) - \beta^{\frac{1}{2}}\sigma_i(\mathbf{x}) \leq y_i \leq m_i(\mathbf{x}) + \beta^{\frac{1}{2}}\sigma_i(\mathbf{x})\},$$

where $i \in \{1, 2, 3\}$, corresponding to area, power and delay metrics in physical space. $\beta$ is a user-defined parameter which determines the impact of $\sigma_i(\mathbf{x})$ on the region. In our implementation, $\beta$ is set to 16 based on the analysis in [27, 120].

As shown in Figure 3.8, the PAL algorithm is an iterative process. A few new points are selected in each iteration, and the GP model is retrained with new training set. Note that the model is supposed to be more and more accurate as more data being sampled. Therefore, the uncertainty region should be smaller and smaller. In order to ensure the non-increasing monotonicity of the uncertainty region while sampling and incorporating the previous evaluations, the uncertainty region of $\mathbf{x}$ in the $(t + 1)$-th iteration is defined as

$$R_{t+1}(\mathbf{x}) = R_t(\mathbf{x}) \cap HR(\mathbf{x}), \tag{3.15}$$

where the initial $R_0 = \mathbb{R}^n$ which is the entire objective space.

The numbers of designs in Pareto-optimal set $P$ and non-Pareto-optimal set $N$ are non decreasing as iteration $t$ increments. Thus, at iteration $t$, the points in $P$ and $N$ keep their classification. Intuitively, if one wants to compare the predicted performance of two designs, two extreme cases, i.e., optimistic prediction $\min(R_t(\mathbf{x}))$ and the pessimistic prediction $\max(R_t(\mathbf{x}))$ of each design, can be applied. If the optimistic prediction of design $\mathbf{x}$ is dominated by the pessimistic prediction of other design $\mathbf{x}'$, then $\mathbf{x}$ is classified as non-Pareto-optimal; And if the pessimistic prediction of design $\mathbf{x}$ is not dominated by optimistic prediction of any other design $\mathbf{x}'$, then $\mathbf{x}$ is classified as Pareto-optimal; A design will remain unclassified if neither condition holds. Figure 3.9

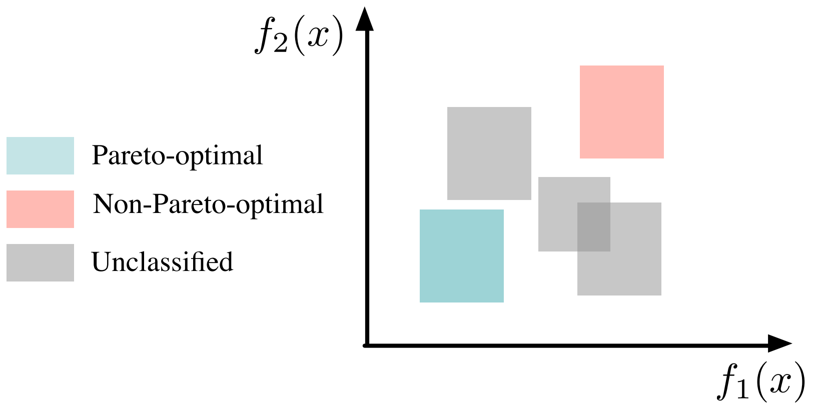


Figure 3.9: An example of classification.

is presented here as an example.

In the implementation, an error tolerance $\delta$ with value 0.001 is applied during classification. The rules for classification can be represented as follows.

$$\mathbf{x} \in \begin{cases} P, & \text{if} \quad \max(R_t(\mathbf{x})) \leq \min(R_t(\mathbf{x}')) + \delta, \\ N, & \text{if} \quad \max(R_t(\mathbf{x}')) \leq \min(R_t(\mathbf{x})) + \delta, \\ U, & \text{otherwise.} \end{cases} \tag{3.16}$$

After classification in each iteration, a new adder design with the largest length of the diagonal of its uncertainty region $R(\mathbf{x})$ is selected for sampling. The value is attached to $\mathbf{x}$ as

$$w_t(\mathbf{x}) = \max_{\mathbf{y},\mathbf{y}' \in R_t(\mathbf{x})} ||\mathbf{y} - \mathbf{y}'||_2. \tag{3.17}$$

Intuitively, Equation (3.17) picks the points which are most worth exploring. Afterward, these designs are going through EDA flow to get the real area, power, and delay numbers, and the GP model will hence be improved with those feedback results.

The entire process is presented in Algorithm 1. It starts with the initialization

**Algorithm 1** Active Learning for Pareto-frontier Exploration

```
Require: Adder architectural design space $E$, GP prior, maximum iteration number $T_{\max}$;
Ensure: predicted Pareto-optimal set $\hat{P}$;
 1: $P \leftarrow \emptyset, N \leftarrow \emptyset, U \leftarrow E$;
 2: Randomly select a small subset $X = \{\mathbf{x}_i\}$ of $E$;
 3: Get true values $Y = \{\mathbf{y}_i | \mathbf{y}_i = \texttt{EDAFlow}(\mathbf{x}_i)\}$;
 4: $S \leftarrow X$;
 5: $R_0(\mathbf{x}) \leftarrow \mathbb{R}^n, \forall \mathbf{x} \in E$;
 6: $t \leftarrow 0$;
 7: while $U \neq \emptyset$ and $t < T_{\max}$ do
 8:     Building GP model with $\{(\mathbf{x}_i, \mathbf{y}_i) : \forall \mathbf{x}_i \in S\}$;
 9:     Obtain $R_t(\mathbf{x}), \forall \mathbf{x} \in E$;
10:     for all $\mathbf{x} \in U$ do
11:         if $\mathbf{x}$ is Pareto-optimal based on Equation (3.16) then
12:             $P.\texttt{add}(\mathbf{x})$, $U.\texttt{delete}(\mathbf{x})$;
13:         else if $\mathbf{x}$ is non-Pareto-optimal based on Equation (3.16) then
14:             $N.\texttt{add}(\mathbf{x})$, $U.\texttt{delete}(\mathbf{x})$;
15:         end if
16:     end for
17:     Obtain $w_t(\mathbf{x}), \forall \mathbf{x} \in (U \cup P) \setminus S$;
18:     Choose $\mathbf{x}' \leftarrow \text{argmax}\{w_t(\mathbf{x})\}$;
19:     $S \leftarrow S \cup \mathbf{x}'$;
20:     $t \leftarrow t + 1$;
21:     Obtain new data $(\mathbf{x}', \mathbf{y}')$ by running EDA flow;
22: end while
23: $\hat{P} \leftarrow P$;
```

(lines 1–6). In each iteration, the GP model is trained with the current training set $S$, and the uncertainty region for each design is obtained (lines 8–9). Then the designs in the $U$ set are classified based on uncertainty regions and classification rules (lines 10–16). After that, the design with the largest uncertainty is sampled and the sampling set $S$ is updated (lines 17–19). The newly sampled design is fed into synthesis tools to get the label which is used for training GP model in the next iteration (line 21). The learning process stops after all adder designs in architectural design space are classified. The prediction is $\hat{P} = P$ (line 23). Suppose $T_{\max}$ is the maximum number of iterations, and $|E|$ is the size of solution set, then the complexity of Algorithm 1 is at most $\mathcal{O}(T_{\max}|E|)$, as maximum size of $U$ can be $|E|$. However, it should be stressed that although there are $T_{\max}|E|$ operations for PAL algorithm, the cost of each operation (which is a simple inference based on the Gaussian Process Regression model) is negligible in comparison to EDA synthesis flow run-time, and we will demonstrate later in Table 3.3 that the total run-time of different approaches are dictated by the number of EDA synthesis flow runs needed in the respective approaches.

## 3.5 Experimental Results

In this section we show the effectiveness of the proposed algorithms and methodologies. First we compare the physical solution space before/after applying PGG algorithm. Then the Pareto frontier obtained by $\alpha$-sweep is presented. Next, we demonstrate the Pareto frontier obtained by active learning, and compare the quality of Pareto frontiers generated by two approaches. Finally, we compare our explored optimal adders against legacy adders.

Since high performance adders are commonly used in CPU architectures which are typically 64 bit, we have mainly presented the results for 64 bit adders to demonstrate the methodology. However, the approach is very general to be used for adders of arbitrary bit-width. The flow is implemented in C++ and Python on Linux machine with 72GB RAM and 2.8GHz CPU. We use Design Compiler [116] (version F-2011.09-SP3) for logical synthesis, and IC Compiler [117] (version J-2014.09-SP5-3) for the placement and routing. `"tt1p05v125c"` corner and Non Linear Delay Model (NLDM) in 32$nm$ SAED cell-library for LVT class [118] (available by University Program) is used for technology mapping. Primary input activity of 0.1 is used along with 1GHz operating frequency for power estimation. Regarding the tool settings, target delays of 0.1$ns$, 0.2$ns$, 0.3$ns$ and 0.4$ns$ are used. Utilization values are set to 0.5, 0.6, 0.7 and 0.8. We used Python based machine learning package scikit-learn [121] for the predictions. Throughout our all experiments, the run time for machine learning predictions is less than a minute.

We relied more on the fidelity of the SAED library rather than accuracy considering that SAED library may not be very realistic as that used in industry. For instance, the FO4 delay for a unit sized inverter for this library in the operating corner is 36$ps$ [1, 122]. So 11 FO4 delay, typically being presented to be the delay for 64-bit adders in literatures [123], is approximately 400$ps$ which is close to the reported delays for 64-bit adders in our work. To further demonstrate the fidelity of this library, we run the Kogge-Stone adders with bit-widths of 8, 16, 32, 64, 128 and 256 through the synthesis flow using this library. Then we normalize the measured delay in terms of FO4 delay, and plot it with bit-width (n) as shown in Figure 3.10. It can be seen that the delay is linear with $\log_2 n$, which is expected for a logarithmic tree adder such as Kogge-Stone adder. So we believe if this algorithmic methodology is applied to more realistic industrial libraries, it can show similar benefit as demonstrated with SAED 32$nm$ library.

To validate the optimality and the hypervolume error of the two learning approaches against the real world solution space, we need to run the logical/physical EDA flow on a large set of adder solutions. Our machine and tool set takes about 5.5 minutes to complete this full flow of a single prefix adder. Therefore, we select a reasonable number (3000) of prefix adder solutions, which eventually took about 300 hours to complete, but

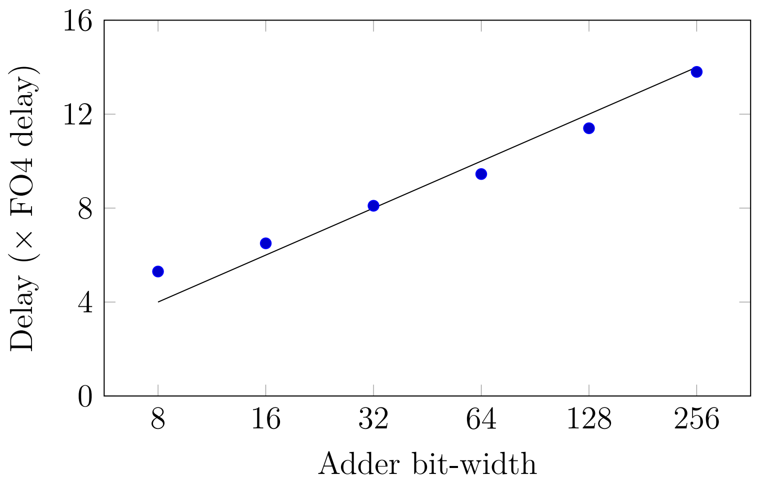


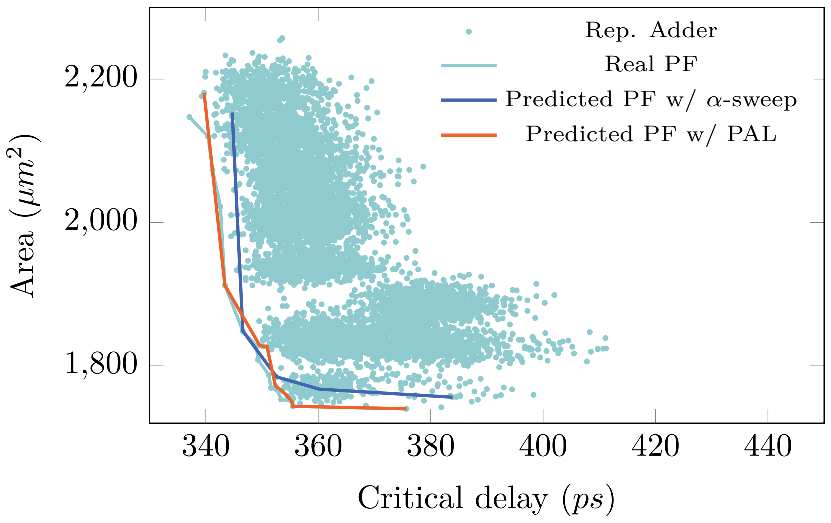
Figure 3.10: Delay values (× FO4 delay) of Kogge-Stone adders with various bit-width.

still a comparatively larger data set in comparison to our training data set. Crucially, those 3000 adders are also sampled in a **Quasi-random** manner in order to represent the entire solution space.

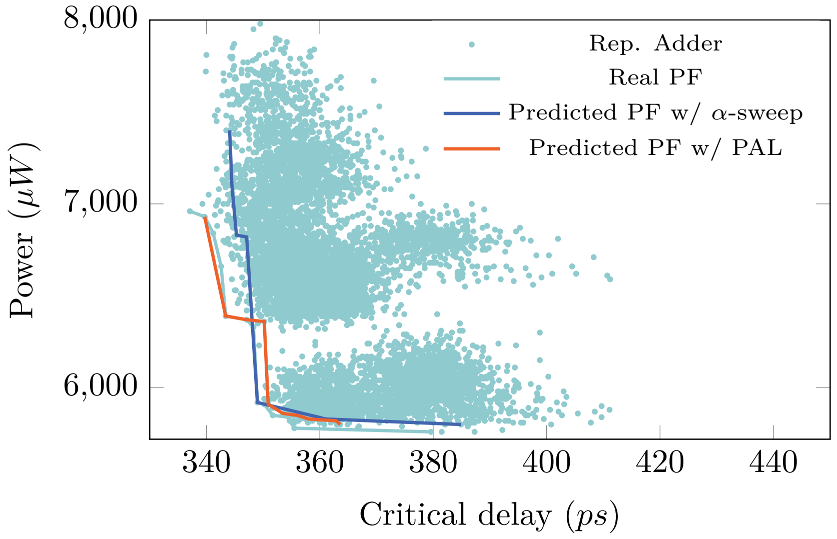
## 3.5.1 Pareto Frontier Predicted by $\alpha$-sweep Learning

In this experiment, we show the effectiveness of our $\alpha$-sweep learning approach. We apply the $\alpha$-sweep method with 15 different $\alpha$ values of $(1000, 0, 100, \frac{1}{100}, 50, \frac{1}{50}, 20, \frac{1}{20}, 10, \frac{1}{10}, 8, \frac{1}{8}, 2, \frac{1}{2}, 1)$, and collect the best 150 solutions for delay-area and delay-power spaces where for each $\alpha$ value, the best 10 architectures with lowest $PD$ or $AD$ values are fed into the logical/physical EDA flow to generate similar Pareto points. Note that $15 + 15 = 30$ learning models have been derived for this for all, but it is very fast as the same training data have been used, and the models are regression based.

Figure 3.11a and Figure 3.11b respectively show the corresponding Pareto frontiers of the $\alpha$-sweep approach and the ground truth Pareto frontiers for the 3000 representative adders. Each dot in the delay-area or delay-power space indicates one adder



(a)



(b)

Figure 3.11: Pareto Frontier: (a) area vs. delay; (b) power vs. delay.

solution after going through the logical/physical EDA flow. We can see that generally the predicted Pareto frontier solutions are fairly close to the real Pareto frontier, with some exceptions. Overall, the proposed approach can effectively achieve near optimal Pareto frontier without affording to spend expensive runtime on every adder. So this

Table 3.1: Comparison of different model accuracies

| Model | MSE | | | Hypervolume error | | |
|---|---|---|---|---|---|---|
| | Area | Power | Delay | Area-Delay | Power-Delay | Area-Power-Delay |
| Original | 0.003 | 0.027 | 0.170 | **0.139** | **0.122** | **0.154** |
| Noisy | 0.024 | 0.951 | 0.711 | 0.168 | 0.148 | 0.162 |

Table 3.2: Pareto frontiers for PAL vs. $\alpha$-sweep [8]

| Objective Hypervolume error | | PAL | $\alpha$-sweep [8] |
|---|---|---|---|
| Area-Delay | average | **0.100** | 0.139 |
| | best | **0.044** | 0.093 |
| Power-Delay | average | **0.109** | 0.122 |
| | best | **0.075** | 0.076 |
| Area-Power-Delay | average | **0.056** | 0.154 |
| | best | **0.039** | 0.125 |

*Notes:* All hypervolume error above are collected from 1000 repeated experiments.

learning based methodology can be readily adopted to achieve Pareto frontiers for much larger solution space which is intractable for exhaustive exploration by conventional design flow.

We have conducted additional experiments to show the impacts of the low accuracy of the machine learning model. The basic idea is to inject random noise in the prediction stage, i.e., additional Gaussian noise is added into the predicted value. The accuracy will be lower than original results. Then we explore the Pareto frontier based on the noisy prediction. Generally, the quality of the final Pareto frontier is worse than original model. The comparison of Pareto frontier quality is presented in Table 3.1.

### 3.5.2 Comparison of the Quality of Pareto-Frontier between PAL and $\alpha$-sweep

We implement PAL to predict Pareto-optimal designs in both two-dimensional design spaces which are area-delay space and power-delay space, as well as three-dimensional space which is area-power-delay space. The results are compared with those of [8]. The initial input set for both area-delay and power-delay is of size 250, which are randomly selected from the exhaustive design space. The curves of Pareto frontiers for two-dimensional spaces are shown in Figure 3.11. The hypervolume of area-delay Pareto frontiers are calculated with reference point (max(delay), max(area)). Similarly, The hypervolume of power-delay Pareto frontiers are calculated with reference point (max(delay), max(power)). Note that the unit for delay is nanosecond ($ns$) when calculating the hypervolume. It should be stressed that there is a sort of randomness in both $\alpha$-sweep and PAL algorithm. For $\alpha$-sweep, the training set is selected randomly. On the contrary, the initial set in PAL is randomly selected (line 2), thereby may result in different outputs. So *the experiments are conducted for 1000 times such that the general performance is reflected.* The comparison between two approaches are shown in Table 3.2. Comparing the hypervolume error of Pareto frontier obtained by PAL and $\alpha$-sweep, it can be seen that PAL achieves better performance in predicting Pareto frontier in all design spaces, including area-delay, power-delay, and area-power-delay spaces.

Table 3.3: Comparison of runtime with single machine among different approaches

| Method | #INIT | #AS | #VERI | #Total | Runtime (mins) | | |
|---|---|---|---|---|---|---|---|
| | | | | | EDA | Modeling | Total |
| Exhaustive | 10000 | - | - | 10000 | 55000.0 | - | 55000.0 |
| $\alpha$-sweep | 2500 | - | 150 | 2650 | 14575.0 | 20.0 | 14595.0 |
| PAL | 700 | 10 | 290 | 1000 | 5500.0 | 2.0 | 5502.0 |

*Notes:* The designs in "#INIT" and "#VERI" can be synthesized in parallel. The number of designs in each category is collected from 1000 repeated experiments.

### 3.5.3 Runtime Comparisons among Exhaustive Approach, $\alpha$-sweep and PAL

There are three factors that will affect the runtime: (i) the total number of EDA synthesis runs required; (ii) Among all these required EDA synthesis runs how many of them can be parallelized; (3) The runtime of the training process in machine learning model. All these details are recorded in Table 3.3. The 'INIT' represents the set of training data in the $\alpha$-sweep and the initial set in PAL, which can be parallelized because all the points are obtained in advance. The 'AS' represents the set of designs which are actively sampled during the learning process, which cannot be parallelized. The $\alpha$-sweep approach does not involve active sampling, so the 'AS' set is none here. The 'VERI' represents the set of designs which are predicted to be Pareto-optimal. We should run EDA synthesis flow to get the real PPA values of these designs to extract the Pareto-frontier. This set of designs are obtained after the learning process stops, so the EDA synthesis runs on these designs are also conducted offline, which can be parallelized. Each EDA synthesis run takes about 5.5 minutes.

Then we can compare the total runtime of different exploration methodologies. For exhaustive exploration, all the prefix adders should be fed into EDA tools for synthesizing to obtain the value of each metric, which is extremely time-consuming.

There is no training, additional sampling, verification. The total runtime cost involves EDA flows of all the designs in the design space. The Pareto frontier can be extracted from the results, whose runtime is much less than synthesizing and can be neglected. The total runtime is

$$T_{exh} = \frac{5.5 \times \#\text{INIT}}{\#\text{Machines}}. \tag{3.18}$$

Since the entire solution space is so huge that one can hardly run all of them, in our experiment, we sample representative 10K designs by random sampling. The total runtime of synthesizing is about 55000 minutes with single machine. It should be noted that the entire solution space is much more than 10K.

In the exploration by $\alpha$-sweep, not all adders in the design space are needed for synthesizing. The total runtime is

$$T_{\alpha} = \frac{5.5 \times (\#\text{INIT} + \#\text{VERI})}{\#\text{Machines}} + \text{Modeling time}. \tag{3.19}$$

In our experiment, we select 2500 of the designs out of those 10K designs by random sampling to build the model, including training and testing phases. It takes about 1.5 minutes to build the model and make predictions. When exploring in area-power-delay design space, 150 designs on average in the design space are predicted to be Pareto-optimal. So on average 2650 designs are needed. The runtime for synthesizing is 14575 minutes. Note that in terms of learning models, the $\alpha$-sweep method needs to build $15 \times 15 = 225$ models since $\alpha_1$ and $\alpha_2$ both have 15 values to choose from.

Similarly, the runtime of PAL can be calculated by

$$\begin{aligned} T_{PAL} = & \frac{5.5 \times (\#\text{INIT} + \#\text{VERI})}{\#\text{Machines}} + 5.5 \times \#\text{AS} \\ & + \text{Modeling time.} \end{aligned} \tag{3.20}$$

The size of initial set is fixed, which is 700. It takes about 4 minutes to build the model and make predictions during the PAL process. When exploring the Pareto-optimal designs in area-power-delay space, 10 designs on average are sampled during PAL. 290 designs on average in the design space are predicted to be Pareto-optimal. In total, 1000 designs are needed on average. The runtime is 5500 minutes with single machine. PAL algorithm needs to build $N$ models where $N$ is the number of iterations in PAL. In our implementation, the maximum iteration is set to 20. It can be observed that the active learning approach outperforms the $\alpha$-sweep learning in terms of both the quality of Pareto frontier and the number of EDA flow runs.

Note that all the runtime calculations are based on single machine. However, the EDA synthesis runs in all three flows can be distributed to multiple machines if available, except the adders sampled during active learning, which (10 on average in our experiments) is very less in comparison to the total number of the synthesis runs. So PAL can get a significant speedup over $\alpha$-sweep and exhaustive approach with single machine and multiple machines.

### 3.5.4 Different Sampling Strategies in PAL

In the sampling stage of PAL, the number of instances to be sampled has impact on the runtime since the EDA flow is required to obtain the real value for area, power and

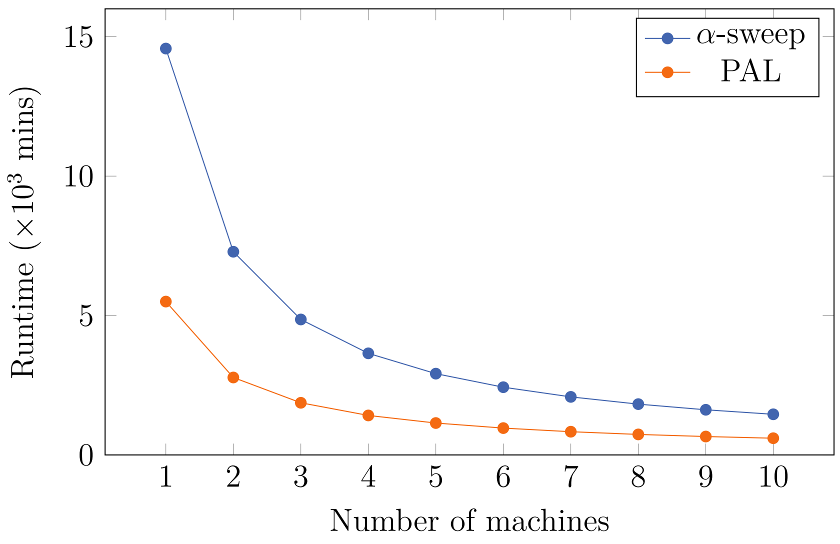
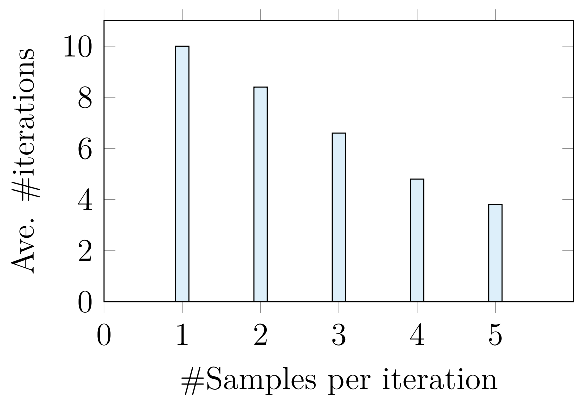


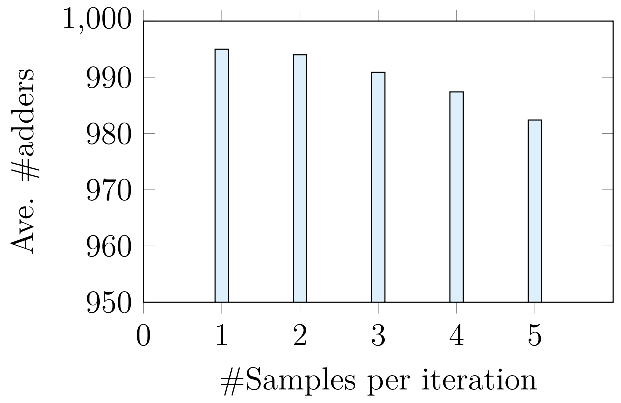
Figure 3.12: Comparison of runtime with different number of machines.

delay. The less instances we sample in each iteration, the more iterations are needed to ensure the PAL process converge, which is more likely to result in less samples in total. The more instances we sample in each iteration, the less iterations are needed. However, the total number of sampled instances would be large. In practice, the runtime cost of running EDA flow can be reduced by parallel execution if there are multiple licenses available. In this section, we explore the effect of different sampling strategies in terms of the total runtime and the quality of Pareto frontier in practical scenarios.

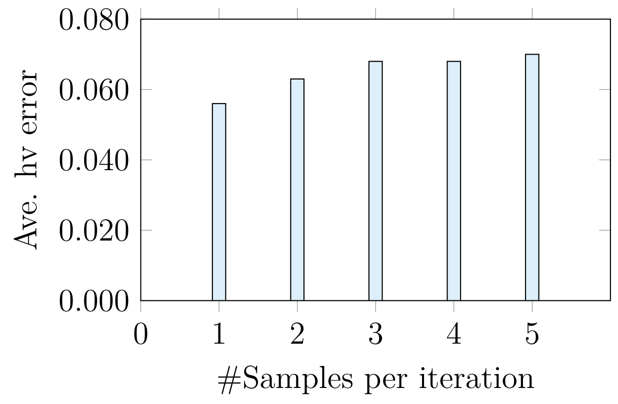
The results for different sampling strategies are listed in Figure 3.13. Since the EDA flow for synthesis, placement and routing takes up the most significant part of the total runtime cost, the key factor is the number of adders which needs to be through EDA tool flow. If we have multiple machines available for the EDA tool flow, the runtime is determined by the total number of iterations as long as the number of samples does not exceed the number of machines. From the result, it can be seen that we can obtain the Pareto-frontier with comparable quality, using less runtime.



(a)



(b)



(c)

Figure 3.13: Comparison among different number of samples per iteration.

Note that when the sample size increases from 1 to 5, the average hypervolume error increases from 0.056 to about 0.070, which is still less than 0.154 (average hypervolume error achieved in $\alpha$-sweep approach). Therefore, batch sampling can not only take care of parallel synthesizing but also achieve better quality for Pareto frontier than $\alpha$-sweep, which can also show the advantages of the PAL.

### 3.5.5 PPA Comparison

Finally, we compare our explored adders against DesignWare adders, legacy adders, such as Kogge-Stone, Sklansky, as well as a state-of-the-art adder synthesis algorithm in Table 3.4. Since our approach generates numerous solutions, it is not feasible to perform a one-to-one comparison. Instead for each of the solution points in regular adders and [124], we have picked the Pareto points from our solution set which are able to excel them in all metrics. For instance, $P_1$ could provide around 8$ps$ better delay with respectively 14% and 12% lesser area and power over Kogge-Stone adder. The DesignWare adders are synthesized from behavioral description of adder (Y = A + B) with the 16 configurations of tool settings (Combination of 4 target delay and 4 utilization values). We pick the one with best delay, denoted by "DesignWare" in Table 3.4. The same pareto point $P_1$ dominates that solution by providing around 7.5$ps$ better delay, 14% lesser area, and 15% lesser energy. For [124] we pick the best delay solution. Note for a fixed $mfo$, [124] can give prefix network with smaller size, but this approach only provides a limited set of prefix structures. As a result, it is hard for [124] to explore the full physical design space of adders by machine learning. It should be stressed that [124] beats the custom adders implemented in an industrial design, and our methodology is able to excel the adders generated by the algorithm

Table 3.4: Comparison with other approaches for 64 bit adders

| Method | Delay ($ps$) | Area ($\mu m^2$) | Energy ($fJ$/op) |
|---|---|---|---|
| DesignWare | 346.5 | 2531.3 | 8160 |
| **Ours** ($P_1$) | 339.0 | 2180.8 | 6930 |
| Kogge-Stone | 347.9 | 2563.7 | 8780 |
| **Ours** ($P_1$) | 339.0 | 2180.8 | 6930 |
| Sklansky | 356.1 | 1792.5 | 6100 |
| **Ours** ($P_2$) | 353.0 | 1753.0 | 5900 |
| [124] | 348.7 | 1971.4 | 6980 |
| **Ours** ($P_3$) | 343.0 | 1912.6 | 6390 |

presented in [124].

## 3.6 Summary

In this chapter, a novel methodology of machine learning guided design space exploration for high-performance prefix adders is presented. We have successfully demonstrated the effectiveness of our learning models, developed by training with quasi-random sampled data and features encapsulating architectural and tool attributes. Moreover, an active learning approach is applied to ease the demand of labeled data and achieves even better Pareto frontier, and provide a remarkable performance vs. power vs. area Pareto frontier over a large representative solution space. To the best of our knowledge, this is the first work to bridge the gap between architectural and physical solution space for parallel prefix adders.

# Chapter 4

# Graph Learning for Testability Analysis

Graphs play a vital role in EDA since they are natural representations for fundamental objects such as netlists and layout. Many problems have previously been solved using graph processing, such as partitioning [125], layer assignment [126], multiple patterning layout decomposition [50, 53] and testability analysis [127]. However, there are few DL-based solutions proposed for graph-based problems in EDA [128]. One reason is that it is not straightforward to generalize CNNs from processing regular grid-based data input to processing graphs. Recently, a number of studies from the DL community have shown how to adopt learning models on graphs, most notably the graph convolutional network (GCN) approach [129, 2, 47]. As shown in the example in Figure 4.1, the essential idea is to obtain an embedding for each node by aggregating information from a node's local neighborhood iteratively such that the node attributes and structural information are encoded. The embeddings can then be leveraged as features in the machine learning tasks.

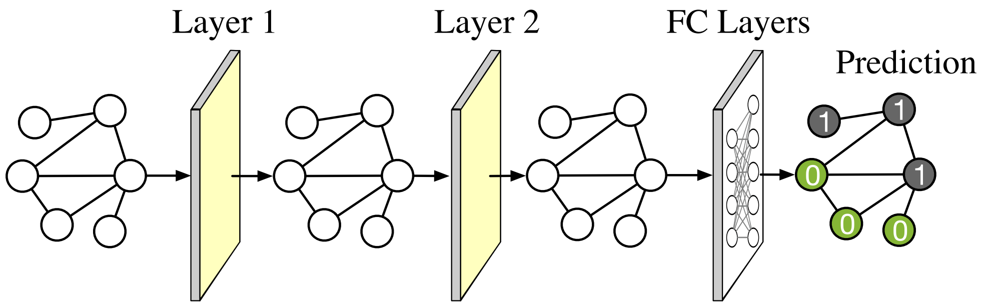


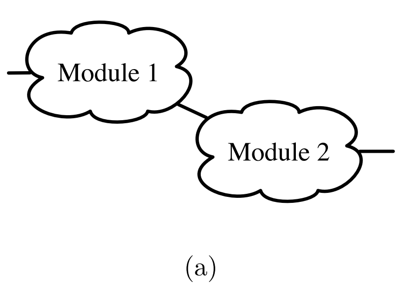
Figure 4.1: Network for node embedding and classification. Layer 1 and Layer 2 generate node embeddings. Nodes are classified through fully-connected (FC) layers.

Circuits testing is a fundamental problem in the design flow, which aims to test each device within reasonable cost and to screen out the parts that may contain defects. However, the difficulty and cost of testing increases as the technology node scales down, which results in more and more defects can be missed and test quality degrades. The reason is that circuits are designed only for functionality and few considerations are given to test. Therefore, these problems lead to the development of design-for-testing (DFT) techniques to improve the testability of circuit designs. Testability is used to measure the effort or cost of testing a logic circuit. Analyzing the testability can help to identify areas of poor testability, and hence can guide to improve the testability.

In this chapter, a high-performance GCN model is proposed for tackling EDA problems with inputs structured as graphs. With a netlist represented as a graph, the GCN model can generate the embedding for each node automatically using the aggregators and encoders, considering both node attributes and graph structural information. By selecting the aggregators properly and leveraging efficient GPU computation, the GCN model is scalable to process a graph containing millions of nodes and edges efficiently. The proposed GCN model is applied to testability analysis for improving circuits testability, which is cast as an imbalanced classification problem and an observation point insertion problem. The proposed GCN model provides fast and accurate prediction, and iteratively inserts observation points to improve testability effectively. The main contributions are summarized as follows:

- A methodology using GCNs for netlist representation and modeling is proposed.
- A parallel training scheme with multiple GPUs and a fast inference scheme are presented for efficient GCN training and inference, which can scale to millions of standard cells.
- A GCN classifier is trained to predict observation point candidates in a netlist, and the proposed GCN classifier is integrated in an iterative observation point insertion process.
- Experimental results indicate the GCN outperforms conventional machine learning models in terms of classification accuracy, and the proposed flow can achieve superior testability results to conventional testability analysis tool on industrial designs.

The rest of this chapter is organized as follows. In Section 4.1, preliminary material about test point insertion is introduced. The proposed GCN model is presented in Section 4.2. Section 4.3 describes how to integrate the GCN classifier to observation points insertion flow. The experimental results are reported in Section 4.4.



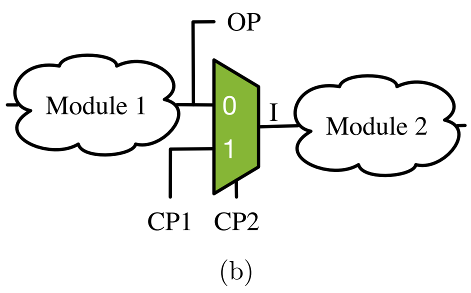


Figure 4.2: (a) Original circuit. Module 1 is unobservable. Module 2 is uncontrollable; (b) Insert test points to the circuit. Set (CP1, CP2) to (0, 1) and (1, 1) will set line I to 0 and 1, respectively; Set CP2 to 0 is the normal operation mode.

## 4.1 Preliminaries

### 4.1.1 Test Points Insertion

Test point insertion (TPI) is a broadly used approach in DFT to modify a circuit and improve its testability, which involves adding extra control points (CPs) or observation points (OPs) to the circuit. An example of TPI is given in Figure 4.2. CPs can be used for setting signal lines to desired logic values, while OPs are added as scan cells to make a node observable. The ultimate goal of TPI is to achieve high fault coverage with less performance degradation. The approach investigated in this chapter is generic and can be applied to both CPs insertion and OPs insertion.

### 4.1.2 Problem Formulation

In this work, we focus on applying observation points insertion (OPI) to improve the testability of a design. From the perspective of a machine learning model, finding the location where the observation points should be inserted in a circuit can be cast as a binary classification problem. For each gate in a design, the problem is whether to add an observation point on the output port or not. If historical data on a sufficient number of designs can be obtained, a classifier can be trained and applied to other designs. Considering that the netlist can be easily represented as graph, GCN is an appropriate approach for this application.

**Problem 1 (Observation Points Insertion)** *Given a set of netlists with all the nodes labeled as either difficult-to-observe or easy-to-observe. The objective is to train a classifier and adopt it to find a set of locations where the observation points should be inserted, which can maximize fault coverage and minimize observation points number and test pattern number.*

## 4.2 GCN for Node Classification

### 4.2.1 Dataset Generation

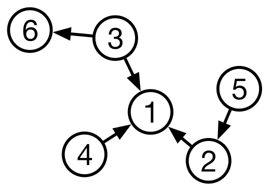
A netlist is represented as a directed graph in which each node corresponds to a cell and each edge is a wire. The source nodes and sink nodes correspond to primary input and primary output, respectively. Each node has an attribute which is a four dimensional feature vector $[LL, C0, C1, O]$. $LL$ denotes logic level of the corresponding gate. $[C0, C1, O]$ are SCOAP measurements [130], which correspond to controllability-0, controllability-1 and observability, respectively. Every node also has a binary label. '0' (negative) means easy-to-observe and '1' (positive) means difficult-to-observe. Labels can be obtained from commercial DFT tools. Given the graphs with node attributes and node labels, a GCN model can be trained, which will be introduced later.

### 4.2.2 Node Embedding Structures

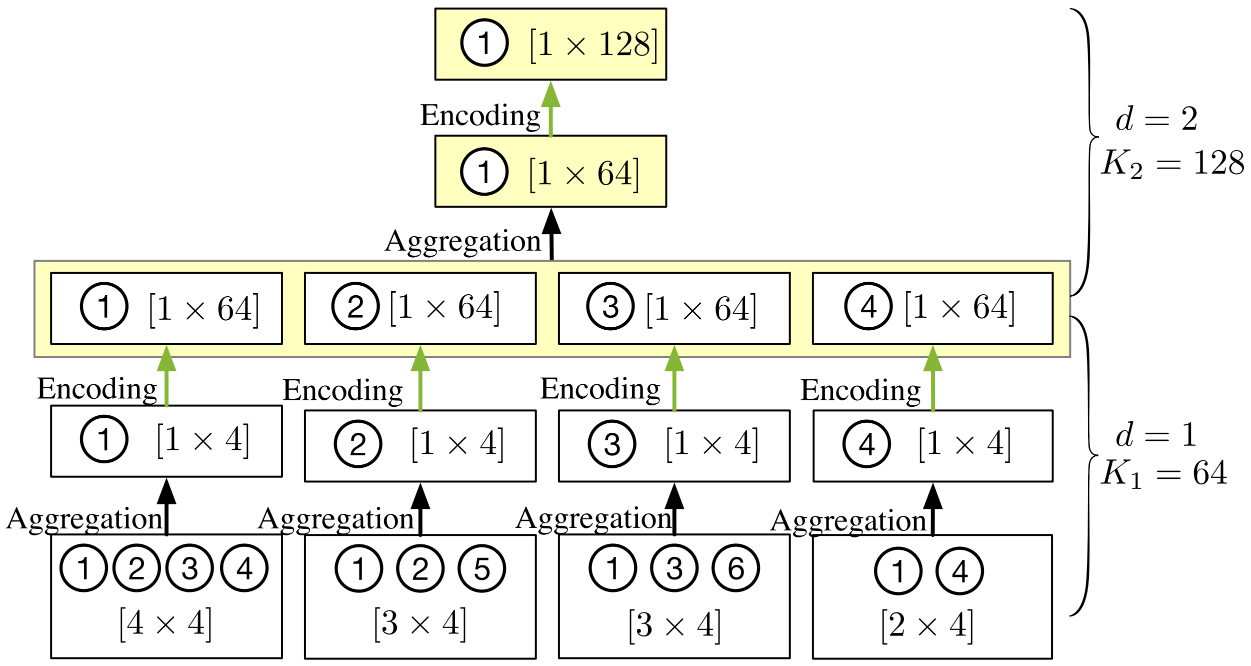
To classify a node in the graph, the neural network first generates its node embedding which is not only based on its own attributes but also the structural information of the local neighborhood. Then, a classification model takes the node embedding as input and predicts a label. To achieve this, three kinds of modules are included in our GCN model, which are aggregator, encoder and classifier. Aggregators and encoders are used to generate the node embeddings by exploiting node attributes and the neighborhood information. The classifier predicts the label for each node in the graph based on its embedding.

Next, we introduce how the node embedding is generated using aggregators and encoders. Essentially, an aggregator or an encoder can be interpreted as a layer of the GCN. Each of them performs a specific operation on the node. An aggregator gathers the feature information from the node's neighbors using an aggregation function $Agg(\cdot)$. An encoder is applied to propagate information between different layers using a weight matrix. The embedding computation process, i.e., *aggregation* and *encoding*, is performed iteratively. Fully-connected layers are used as classifier which takes the node embedding as input, and predicts the label for the node.

Suppose that the network is trained and all the weights are obtained. Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and node attributes $\{\boldsymbol{x}^{(v)} : \forall v \in \mathcal{V}\}$, the node embeddings $\{\boldsymbol{e}^{(v)} : \forall v \in \mathcal{V}\}$ are generated as in Algorithm 2. Since the node embedding is expected to aggregate the information in local neighborhood, a depth $D$ is specified to indicate the "radius" of the neighborhood region of a node. The initial representation $[LL, C0, C1, O]$ is set as the node attributes (line 1). There are two loops involved. In each step of the outer loop, the representation of each node is updated through aggregation and



(a)



(b)

Figure 4.3: An illustration to compute the embedding for a node with $D = 2$. (a) Graph; (b) Procedure to compute the embedding for node 1.

encoding. More specifically, in the $d$-th iteration of the outer loop, every node first aggregates information from its neighbors using aggregation function $Agg(\cdot)$ which takes the representations of node $v$ and its neighbors generated at $(d-1)$-th iteration as input, and generates a new representation for node $v$, denoted by $\boldsymbol{g}_d^{(v)}$ (line 4). We use a weighted sum function as the aggregation function. Assume predecessors (PR) and successors (SU) have different weights. The aggregation $Agg(\cdot)$ can be formulated explicitly as:

$$\boldsymbol{g}_d^{(v)} = \boldsymbol{e}_{d-1}^{(v)} + w_{pr} \times \sum_{u \in \mathrm{PR}(v)} \boldsymbol{e}_{d-1}^{(u)} + w_{su} \times \sum_{u \in \mathrm{SU}(v)} \boldsymbol{e}_{d-1}^{(u)}, \tag{4.1}$$

where $w_{pr}$ and $w_{su}$ are weights for predecessors and successors, respectively, and they are the same in each step of outer loop. Next, a non-linear transformation is performed to encode the aggregated representation using a weight matrix $\boldsymbol{W}_d \in \mathbb{R}^{K_{d-1} \times K_d}$ and an activation function $\sigma(\cdot)$ (line 5). $K_d$ is the dimension of the embedding after $d$-th step and $K_0$ is 4 which is the initial attribute dimension. A concrete example is shown in Figure 4.3 which illustrates the procedure of computing node embedding with $D = 2$. Essentially, after $d$ iterations, the embedding of a node combines the information of its $d$-hop neighborhood.

When maximum depth $D$ is reached, the final embeddings are obtained and fed to the fully-connected layers for classification. Parameters that need to be trained include $w_{pr}$, $w_{su}$, $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_D$ and parameters in FC layers. All the parameters in the network can be trained end-to-end.

Different from other transductive approaches which cannot generalize to unseen graphs [129, 46], the entire classification procedure for each node is only based on its neighborhood information and the learned parameters and can be shared across different

**Algorithm 2** Node Embedding Computation

```
Require: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; node attributes $\{\boldsymbol{x}^{(v)} : \forall v \in \mathcal{V}\}$; Search depth $D$; non-linear
    activation function $\sigma(\cdot)$; Weight matrices $\boldsymbol{W}_d$ of encoders $\boldsymbol{E}_d, d = 1, ..., D$;
Ensure: Embedding of for each node $\boldsymbol{e}_D^{(v)}, \forall v \in \mathcal{V}$.
1: $\boldsymbol{e}_0^{(v)} \leftarrow \boldsymbol{x}^{(v)}, \forall v \in \mathcal{V}$;
2: for $d = 1, ..., D$ do
3:     for all $v \in \mathcal{V}$ do
4:         Compute $\boldsymbol{g}_d^{(v)}$ based on Equation (4.1);          ▷ Aggregation
5:         $\boldsymbol{e}_d^{(v)} \leftarrow \sigma(\boldsymbol{W}_d \cdot \boldsymbol{g}_d^{(v)})$;          ▷ Encoding
6:     end for
7: end for
```

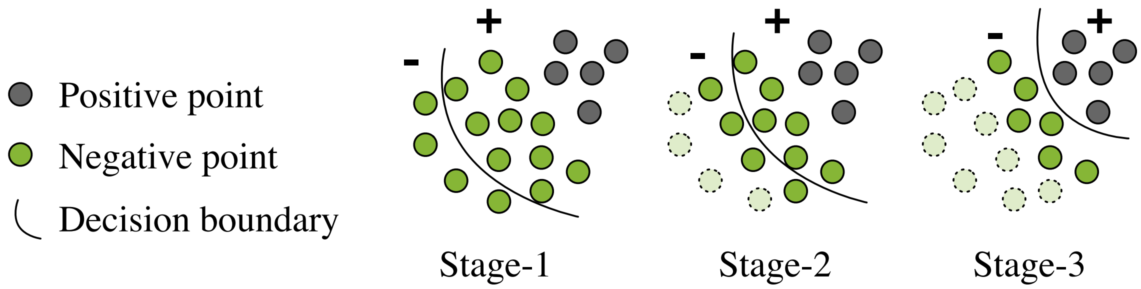


Figure 4.4: An example of three-stage GCN classification.

graphs.

### 4.2.3 Multi-stage Classification

For a typical design, it is common to have many more negative nodes than positive nodes, which is not desireable for training machine learning models. Training a single classification model can lead to poor overall performance since significant bias would be introduced towards the majority class. To tackle this imbalance issue, we developed a multi-stage GCN for this problem. In each stage, a GCN is trained and only filters out negative cases with high confidence, and passes the remaining nodes to the next stage, which is illustrated in Figure 4.4. This is achieved by imposing a large weight on the positive nodes such that the penalty of misclassifying them would be large. In this way, most positive points remain on the right side of the decision boundary until negative points are substantially reduced. After a few stages, the remaining nodes should become relatively balanced and a network can make the final predictions.

### 4.2.4 Efficient Training and Inference

#### 4.2.4.1 Inference

Making the GCN model scalable to large graphs is a critical problem, especially because fast inference is desired. The computation shown in Algorithm 2 is an iterative process. However, it would be inefficient since the neighborhoods of different nodes may have overlap, thus there are many duplicated computations [2, 47]. Here we introduce another approach to inference computation that enables our GCN to process millions of nodes efficiently. The key idea is to leverage the adjacency matrix of the graph, denoted by $\boldsymbol{A} \in \mathbb{R}^{N \times N}$. $N$ is the total number of nodes in the graph. A matrix $\boldsymbol{E}_d \in \mathbb{R}^{N \times K_d}$ can also be obtained, in which the $v$-th row represents the embedding of node $v$ after the $d$-th iteration, i.e., $\boldsymbol{E}_d[v, :] = \boldsymbol{e}_d^{(v)}$. Take the graph in Figure 4.3a as an example. The weighted sum aggregation in iteration $d$ is equivalent to Equation (4.2).

$$\boldsymbol{G}_d = \boldsymbol{A} \cdot \boldsymbol{E}_{d-1} = \begin{array}{c c} & \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} & \begin{bmatrix} 1 & w_1 & w_1 & w_1 & 0 & 0 \\ w_2 & 1 & 0 & 0 & w_1 & 0 \\ w_2 & 0 & 1 & 0 & 0 & w_2 \\ w_2 & 0 & 0 & 1 & 0 & 0 \\ 0 & w_2 & 0 & 0 & 1 & 0 \\ 0 & 0 & w_1 & 0 & 0 & 1 \end{bmatrix} \end{array} \times \begin{bmatrix} \boldsymbol{e}_{d-1}^{(1)} \\ \boldsymbol{e}_{d-1}^{(2)} \\ \boldsymbol{e}_{d-1}^{(3)} \\ \boldsymbol{e}_{d-1}^{(4)} \\ \boldsymbol{e}_{d-1}^{(5)} \\ \boldsymbol{e}_{d-1}^{(6)} \end{bmatrix} \tag{4.2}$$

Here $\boldsymbol{A} \in \mathbb{R}^{6\times6}$, $\boldsymbol{G}_d \in \mathbb{R}^{6\times K_{d-1}}$, and the $v$-th row is the representation for node $v$ after aggregation in the $d$-th iteration. The inner loop in Algorithm 2 (line 3 – line 6) can be simply formulated as

$$\boldsymbol{E}_d = \sigma(\boldsymbol{G}_d \cdot \boldsymbol{W}_d) = \sigma((\boldsymbol{A} \cdot \boldsymbol{E}_{d-1}) \cdot \boldsymbol{W}_d). \tag{4.3}$$

Then, all the computation can be formulated as a series of matrix multiplications which can be efficiently computed, and duplicated computation can be avoided. One potential issue is the dimension of adjacency matrix $\boldsymbol{A}$ is $N \times N$, which is extremely large and cannot be stored in memory directly. However, we can exploit the fact that the $\boldsymbol{A}$ is a sparse matrix. For every design in our benchmarks, the sparsity is higher than 99.95%. Then $A$ can be represented in a compressed sparse format, e.g., coordinate (COO) format which stores a list of (`value`, `row_index`, `column_index`) tuple. The matrix can be stored in the memory to enable the matrix multiplication. For instance, the COO representation of $\boldsymbol{A}$ in Equation (4.3) is represented as

`value:` $[1, w_1, w_1, w_1, w_2, 1, w_1, w_2, 1, w_2, w_2, 1, w_2, 1, w_1, 1]$
`r_index:`$[1, 2, 3, 4, 1, 2, 5, 1, 3, 6, 1, 4, 2, 5, 3, 6]$,
`c_index:`$[1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 5, 5, 6, 6]$

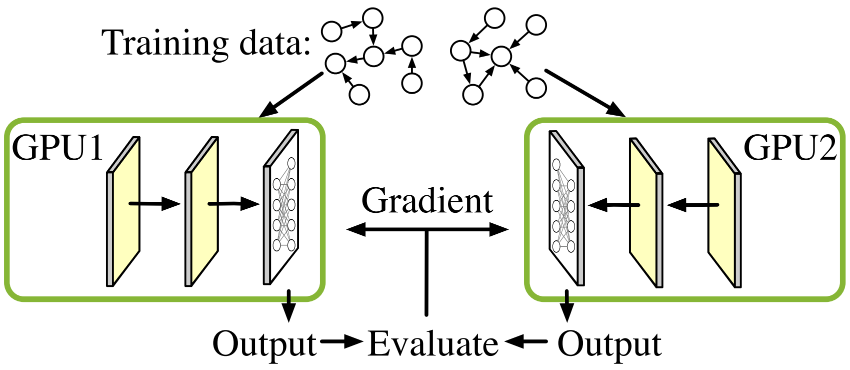


Figure 4.5: Parallel training with multiple GPUs.

where each column is a tuple indicating the value and indices of a non-zero element in the matrix. Moreover, the COO format is very efficient for incremental matrix construction which facilitates graph modifications in our flow.

#### 4.2.4.2 Training

The strategy proposed for inference can also accelerate the GCN training process. In practice, the training set may contain many graphs. A straightforward method is to compose multiple graphs into a single graph, but the memory of a single GPU may not be sufficient to hold all intermediate results in this case. To overcome this bottleneck, we leverage a parallel training scheme with multiple GPUs shown in Figure 4.5. Our scheme can be seen as a variant of a conventional data-parallel scheme. Conventionally, a batch of input data is split into equal chunks and each GPU processes a chunk. With our GCN approach, the input of one graph includes an adjacency matrix and node representation matrix which cannot be split. Therefore, we separate multiple graphs

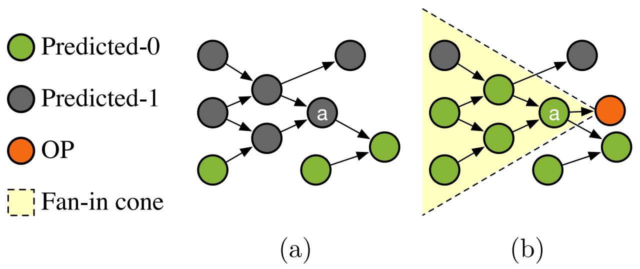


Figure 4.6: An illustration to compute the impact for an OP. (a) Prediction on original graph with 5 positives in fan-in cone of a; (b) Prediction on graph after inserting an observation point with 1 positive. The impact of node $a$ is $5 - 1 = 4$.

into individual graph. Each GPU processes one graph, and all of the output is gathered to calculate the loss and then do back-propagation to update the model.

## 4.3 Iterative OP Insertion

After a multi-stage GCN model is trained, it can identify difficult-to-observe nodes in a netlist, which can be used as guidance for observation point insertion. However, not every difficult-to-observe node has the same *impact* for improving the observability since it is possible that adding an observe point may improve the observability of other nodes in its fan-in cone. In order to minimize the number of insertion points, we must select the observation point locations with largest *impact*. Next we develop a flow to identify which locations are more significant using the trained classifier. We define the *impact* of each location as the positive prediction reduction in a local neighborhood after inserting an observation point. Figure 4.6 gives an example of impact calculation. An iterative flow for points insertion is developed, which is shown in Figure 4.7. In each iteration, every positive prediction is evaluated to get its impact. Finally, a list containing observation points location is obtained. Then they are sorted based on their

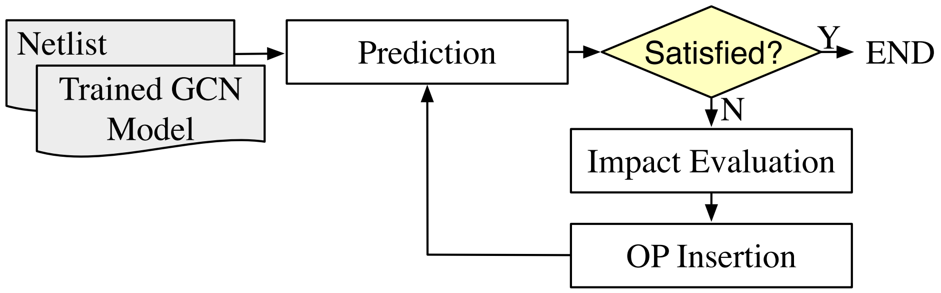


Figure 4.7: Iterative flow for observation points insertion.

impact and select the top ranked locations. Next, we modify the graph and perform inference for prediction. The positive predictions will become the candidates in the next iteration. The exit condition is there are no positive predictions left.

Inserting observation points modifies the netlist, thus the graph should be modified, including the graph structure and node attributes. Inserting one observation point to a target node $v$ corresponds to adding a new node $p$ to the graph and adding an edge from the target node $v$ to new node $p$. Essentially, the adjacency matrix $\boldsymbol{A}$ and initial embedding matrix $\boldsymbol{E}_0$ should be updated. A critical problem in this iterative flow is how to update the graph efficiently. In our flow Figure 4.7, the graph can be updated incrementally. $\boldsymbol{A}$ can be incrementally updated by adding a column and a row, and setting corresponding entries as $w_{pr}$ or $w_{su}$, which can be done efficiently under COO format by appending 3 tuples $(w_{pr}, p, v)$, $(w_{su}, v, p)$ and $(1, p, p)$. $\boldsymbol{E}_0$ is updated by appending attribute of new node $p$ which is set to [0,1,1,0]. Then only the attributes of the nodes in the fan-in cone of the new node should be updated based on SCOAP method [130]. Since this GCN model is inductive, the updated $\boldsymbol{A}$ and $\boldsymbol{E}_0$ are directly fed to the model for prediction.

Table 4.1: Statistics of benchmarks

| Design | #Nodes | #Edges | #POS | #NEG |
|---|---|---|---|---|
| B1 | 1384264 | 2102622 | 8894 | 1375370 |
| B2 | 1456453 | 2182639 | 9755 | 1446698 |
| B3 | 1416382 | 2137364 | 9043 | 1407338 |
| B4 | 1397586 | 2124516 | 8978 | 1388608 |

## 4.4 Experimental Results

The experiments are performed on 4 industrial designs implemented in 12$nm$ technology. Statistics of designs are summarized in Table 4.1. #POS and #NEG indicate the number of difficult-to-observe nodes and easy-to-observe nodes, respectively. The labels are obtained from commercial DFT tools. The GCN is implemented with PyTorch and trained on a Linux machine with 32 cores and 4 NVIDIA Tesla V100 GPUs. The total memory used in the training is 64GB.

### 4.4.1 Node Classification Results

Search depth is an important hyper-parameter that may affect the performance of a GCN. On one hand, increasing search depth can cover a larger neighborhood such that more information can be involved. On the other hand, too large region may lead to over-fitting. We select the search depth by monitoring and comparing the training and testing accuracy among different settings on the search depth. $K_1$, $K_2$ and $K_3$ are set as 32, 64 and 128, respectively. The Rectified linear unit (ReLU) function $\text{ReLU}(x) = \max(x, 0)$ [131] is used as the activation function. Four FC layers are consistent, whose dimensions are 64, 64, 128 and 2. Figure 4.8 shows the record of training accuracy and testing accuracy during learning for 300 epochs with search depth

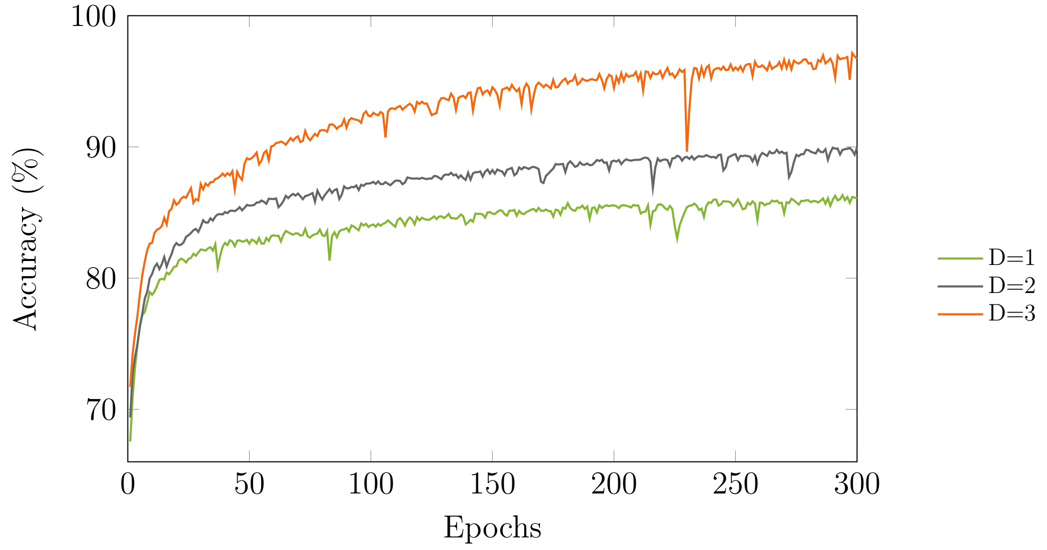


Figure 4.8: Performance with different search depth $D$.

1, 2 and 3. It can be seen that the performance of GCN improves as the search depth increases. Search depth $D = 3$ is used for all the remaining experiments. The entire network consists of 3 aggregators, 3 encoders (with ReLU layer) and 4 FC layers. We use cross-entropy function as loss function and stochastic gradient descent for optimization.

We compare the classification performance between our proposed GCN and various classical machine learning models, including logistic regression (LR), random forest (RF), support vector machine (SVM) and Multi-layer perceptron (MLP). Considering a single classifier may not have good performance on the benchmark that is highly imbalanced, we compare the performance on balanced datasets by sampling a subset within the entire dataset. Since other classical machine learning models require handcrafted features with a fixed dimension, we integrate neighborhood features by collecting the features of the nodes in the fan-in cone and fan-out cone. 500 nodes in fan-in cone and 500 nodes in fan-out cone are collected. Starting from the target node, breadth-first-search is performed to collect the nodes in each cone. Every time a node is visited, the

Table 4.2: Accuracy comparison on balanced dataset

| Design | LR | RF | SVM | MLP | GCN |
|---|---|---|---|---|---|
| B1 | 0.778 | 0.790 | 0.813 | 0.860 | **0.928** |
| B2 | 0.767 | 0.785 | 0.809 | 0.845 | **0.929** |
| B3 | 0.779 | 0.793 | 0.814 | 0.856 | **0.930** |
| B4 | 0.782 | 0.801 | 0.821 | 0.862 | **0.935** |
| Average | 0.777 | 0.792 | 0.814 | 0.856 | **0.931** |

feature of this node is concatenated to the current feature vector. Therefore, the dimension of the feature vector for traditional learning models is $(500 + 500 + 1) \times 4 = 4004$.

For each design, we generate a balanced dataset by using all the positive nodes and sampling the same number of negative nodes randomly. Each time we use three designs for training and the remaining one for testing. For the MLP approach, the configuration of the network is the same as the classifier module in GCN. The accuracy comparison is presented in Table 4.2. GCN achieves 93.1% accuracy on average, outperforming other models on all designs.

### 4.4.2 Visualization of node embeddings with t-SNE

In the proposed GCN, each node needs to go through aggregation-encoding process for three times. Actually, the output vectors of E1, E2 and E3 correspond to the representation vectors for a node after integrating features of the nodes in 1-hop neighborhood, 2-hop neighborhood and 3-hop neighborhood, respectively. Data visualization can facilitate us to justify whether the representation of a node is discriminative or not. The representation vector of each node generated by the GCN is in high dimensional space, which is difficult to inspect directly. Principle component analysis (PCA) and t-SNE

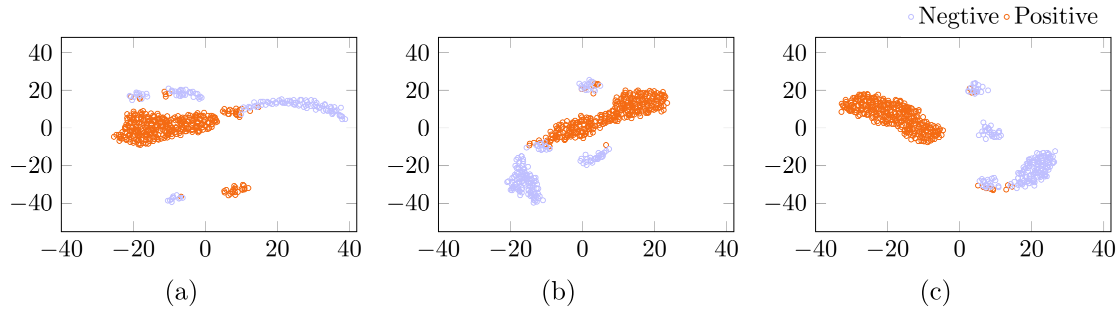


Figure 4.9: Visualization of node embeddings with different search depth $K$. (a) $K$=1; (b) $K$=2; (c) $K$=3.

[132] are widely used techniques for high dimensional data analysis and visualization. In this experiment, we visualize the feature representation obtained from different encoders using t-SNE for 1000 nodes, including 500 positive nodes and 500 negative nodes. The visualization is shown in Figure 4.9. It can be observed that the representations obtained for positive class and negative class becomes more discriminative as search depth increases.

### 4.4.3 Multi-stage Classification

We have shown that a GCN can obtain significantly better performance in distinguishing positive nodes and negative nodes than classical learning models. However, the performance a single GCN is not satisfying if it is directly trained and tested on original imbalanced dataset. To validate the advantage of the multi-stage GCN, we compare the performance between the multi-stage GCN and single GCN. In highly imbalanced classification, *F1-score* is commonly used since accuracy would be misleading. The training and testing schemes are the same as before. Each time we use three designs for training and the remaining design for testing. 3 stages are applied and the prediction

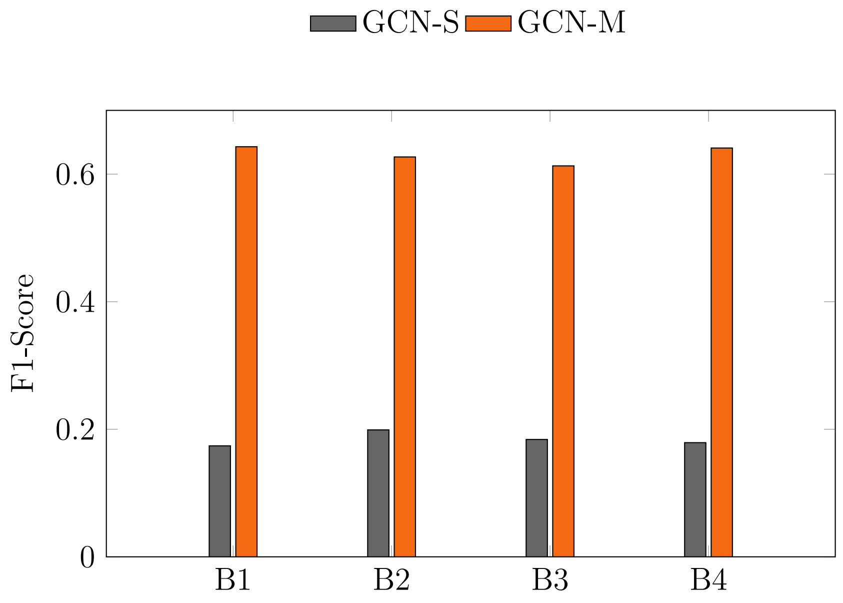


Figure 4.10: F1-Score comparison.

results of each stage are combined to obtain the F1-score on the entire dataset. The results comparison is shown in Figure 4.10. The multi-stage GCN achieves much higher F1-scores than single GCN on all these imbalanced datasets.

### 4.4.4 Scalability vs. State-of-the-art GCN

By taking the advantage of the sparse representation of the adjacency matrix and matrix multiplication-based computation, the proposed GCN can scale to process designs with millions of cells. To demonstrate the scalability of our acceleration strategy, we compare the inference runtime between two approaches on graphs with different sizes, ranging from $10^3$ nodes to $10^6$ nodes. We leverage the released implementation of [2] for comparison, which uses recursion-based computation. The inference runtime comparison between two approaches is shown in Figure 4.11. For a design with 1 million cells, the recursive computation takes more than one hour to complete the inference.

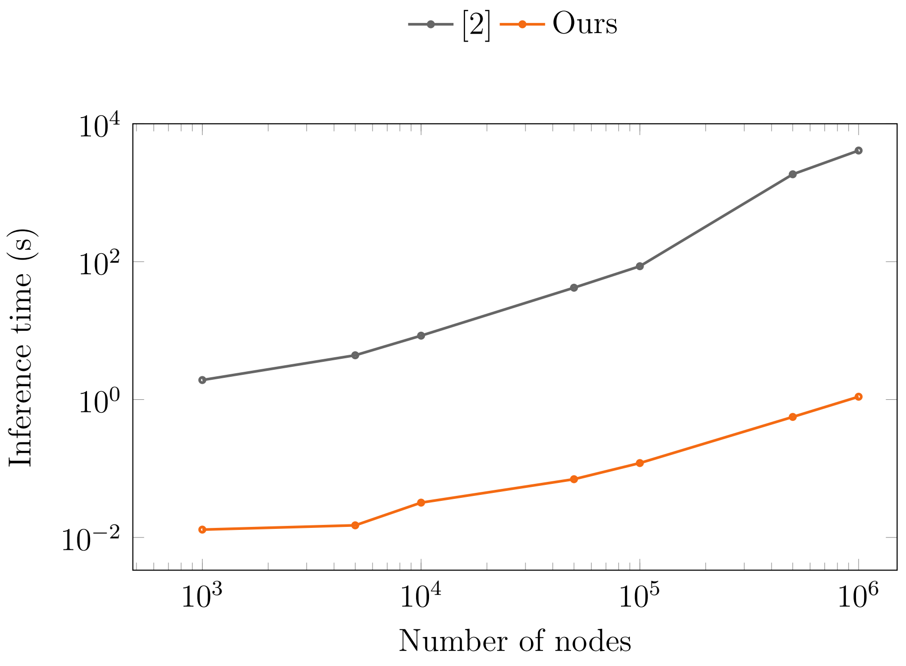


Figure 4.11: Scalability vs. [2].

With our acceleration strategy, the inference runtime is only 1.5 seconds, which is three orders of magnitude speedup compared to [2].

### 4.4.5 Verified in Industrial Testing Flow

Finally we show the testability results of applying the multi-stage GCN and the iterative test points insertion flow. We use the testability analysis results from a commercial testability analysis tool as a baseline. 3 metrics are used for evaluation, including the total number of OPs inserted, the fault coverage and the number of test patterns required. The output list of our proposed iterative GCN-based flow is fed to the same commercial tool to get a test report from which we can get the pattern number and fault coverage for a fair comparison. The results are shown in Table 4.3. Column '#OPs' represents the total number of OPs inserted. '#PAs' represents the number

Table 4.3: Testability results comparison

| Design | Industrial Tool | | | GCN-Flow | | |
|---|---|---|---|---|---|---|
| | #OPs | #PAs | Coverage | #OPs | #PAs | Coverage |
| B1 | 6063 | 1991 | 99.31% | 5801 | 1687 | 99.31% |
| B2 | 6513 | 2009 | 99.39% | 5736 | 2215 | 99.38% |
| B3 | 6063 | 2026 | 99.29% | 4585 | 1845 | 99.29% |
| B4 | 6063 | 2083 | 99.30% | 5896 | 1854 | 99.31% |
| Average | 6176 | 2027 | **99.32%** | **5505** | **1900** | **99.32%** |
| Ratio | 1.00 | 1.00 | **1.00** | **0.89** | **0.94** | **1.00** |

of test patterns. 'Coverage' represents fault coverage. It can be seen that the OPs recommended by the proposed GCN model and iterative flow outperform the industrial tool, achieving 11% reduction on the number of inserted OPs and 6% reduction on test patterns without any degradation on fault coverage.

## 4.5 Summary

In this chapter, a GCN-based methodology is proposed to identify difficult-to-observe points in a netlist, in which GCN shows superior performance to classical learning models. A multi-stage GCN is developed to handle the imbalanced classification problem, which achieves significantly higher F1-score than single GCN model. A parallel training scheme is developed to enable large scale training and a fast inference scheme is proposed to enhance the scalability of the GCN model. Based on the GCN model and an iterative observation points insertion flow, we have achieved better testability on industrial designs compared to a widely used commercial tool.

# Chapter 5

# Unified Mask Optimization

In accordance with the Moore's law, through extreme scaling, the transistor number on a chip has increased exponentially in the last five decades. However, the continued scaling of the transistor feature size has pushed the conventional 193$nm$ wavelength lithography system (Figure 5.1) into its resolution limit, thus the whole semiconductor industry is facing severe manufacturing challenges [133]. To overcome these issues, resolution enhancement techniques (RETs) on layout and mask levels toward better printability and yield are of great importance [134].

Two of the most critical RET stages are layout decomposition and mask optimization. In the first stage, layout decomposition divides target image into several masks so that the coarser pitches on every mask can be manufactured through 193$nm$ wavelength lithography. Otherwise, defects can be formed in the manufactured devices. An example is shown in Figure 5.2. Depending on the total mask number available, the problem is also called double patterning layout decomposition (two masks) or triple patterning layout decomposition (three masks). The diffraction effect of the light cannot be ignored since the size of the patterns on layout is comparable with the wavelength of the

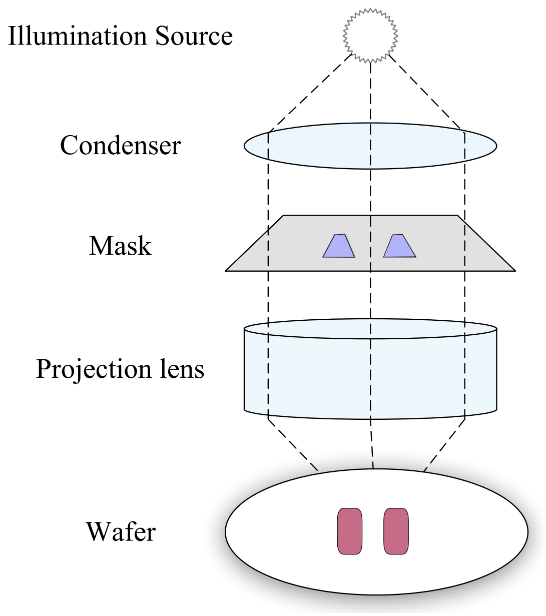


Figure 5.1: An example of a typical lithography system.

lithography light source, which may result in distortion of the final on-wafer image. In mask optimization, e.g., optical proximity correction (OPC), each mask is refined to compensate the diffraction effect of the light in advance to ensure the high quality of on-wafer image, as demonstrated in Figure 5.3. Finally, all optimized masks go through lithography process separately to generate corresponding printed image, and all printed images are combined together to generate the target patterns.

In emerging technology nodes, the conventional two-stage flow (i.e., layout decomposition followed by mask optimization) cannot achieve good printability on their own. The reasons are two-fold. (1) The layout decomposition and mask optimization are separated from each other and each problem is solved independently, which may lose a global view. (2) Due to the inconsistency between the objectives of the two stages, decomposed results with identical quality may cause diverse printed image qualities after mask optimization. That is, the layout decomposition is based on simple design

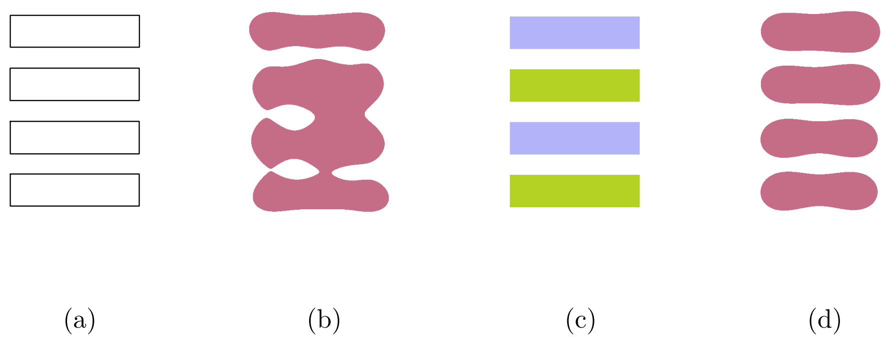


Figure 5.2: An example of double patterning lithography. (a) Without double patterning; (b) Printed image on the wafer without double patterning. Defects are formed; (c) With double patterning. Two masks labeled with different colors; (d) Printed image on the wafer with double patterning. No Defects are formed.

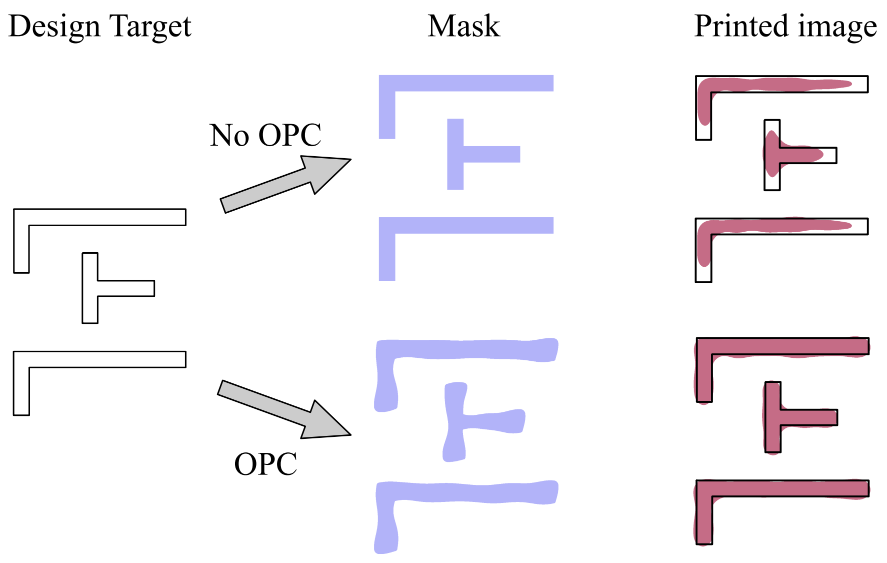


Figure 5.3: Performing OPC leads to better printed image on the wafer.

or coloring rules, which are just coarse regression of complicated lithography model; while the mask optimization is verified by accurate and sophisticated lithography simulation. Figure 5.4 gives an example on such inconsistency. Given the identical target, two different layout decomposition results are found (LD stage in the figures), and both

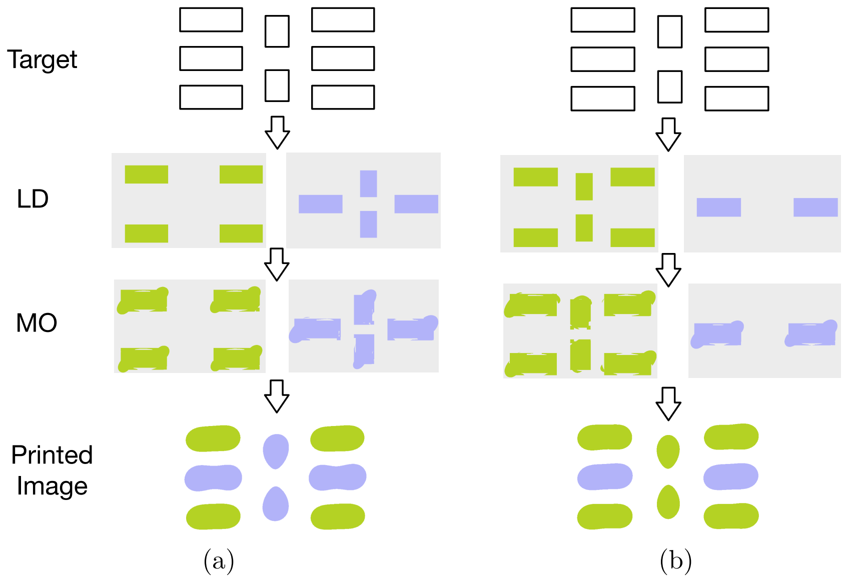


Figure 5.4: Same quality layout decompositions (LD) can achieve different EPE violation number after mask optimization (MO): (a) Solution 1 with #EPE violation = 3; (b) Solution 2 with #EPE violation = 1.

of them satisfy all design rules and coloring rules. After the mask optimization (MO stage in the figures) on each mask, however, it can be observed that the qualities of the printed images are diverse: Figure 5.4a has three EPE violations, while Figure 5.4b has only one EPE violation. Therefore, there is an increasing need to bridge the gap between layout decomposition and mask optimization by a unified design framework.

In this chapter, we propose a unified optimization framework which aims at solving layout decomposition and mask optimization simultaneously. Combining the two processes together leads to a larger solution space, which has potential to obtain a higher quality mask design. In addition, through effective pruning techniques, our framework can avoid exhaustive mask optimization on all layout decomposition solutions, therefore the overall efficiency can be significantly improved. The main contributions of this work are listed as follows.

- To the best of our knowledge, this is the first framework handling multiple patterning layout decomposition and mask optimization simultaneously.
- We propose a unified problem formulation and develop a gradient-based optimization approach, while process variation issue is studied to guarantee mask robustness.
- We further apply a set of discrete optimization techniques (e.g., semi-definite programming, randomized rounding, and pruning) to avoid being stuck in local optimum.
- The experimental results verify the effectiveness of the proposed framework.

The rest of the chapter is organized as follows. Section 5.1 introduces lithography models and evaluation criteria. Section 5.2 formulates the problem mathematically and describes algorithmic details in our framework. Section 5.3 lists the experimental results to support our methodologies, followed by a discussion in Section 5.4 and Section 5.5 summarizes this chapter.

## 5.1 Preliminaries

In this section, we provide preliminaries on lithography models, and then introduce the evaluation criteria. For convenience, notations used in this work are listed in Table 5.1. Note that in this work we focus on DPL scenario, but the problem formulation and the corresponding methodologies can be extended to triple patterning counterpart.

Table 5.1: Notations used in this work.

| | |
|---|---|
| $\boldsymbol{Z}_t$ | Target image |
| $\boldsymbol{Z}_1$, $\boldsymbol{Z}_2$ | Binary images under the nominal condition |
| $\boldsymbol{Z}$ | Printed image under the nominal condition |
| $\boldsymbol{Z}^{(n)}$ | The printed image under the $n$-th process condition |
| $\boldsymbol{M}_1$, $\boldsymbol{M}_2$ | Output masks |
| $\boldsymbol{P}_1$, $\boldsymbol{P}_2$ | Unconstrained variables |
| $\boldsymbol{I}_1$, $\boldsymbol{I}_2$ | Aerial images under the nominal condition |
| $\boldsymbol{I}_1^i$, $\boldsymbol{I}_2^i$ | Aerial images under the $i$-th process condition |
| $\boldsymbol{H}$ | A set of optical kernels $\{\boldsymbol{h}_1, \ldots, \boldsymbol{h}_K\}$ |
| $\boldsymbol{H}^*$ | The conjugate transpose of $\boldsymbol{H}$ |
| $\odot$ | Element-wise matrix multiplication operator |
| $\otimes$ | Convolution operator |

### 5.1.1 Forward Lithography Models

Two models are needed to transform mask patterns into printed image: optical lithography model and photo resist model. First, an aerial image $\boldsymbol{I}$ is generated by convolving the mask $\boldsymbol{M}$ with a set of optical kernels [135], which is represented as

$$\boldsymbol{I} = f_{optical}(\boldsymbol{M}) = \sum_{k=1}^{K} w_k \cdot |\boldsymbol{M} \otimes \boldsymbol{h}_k|^2, \tag{5.1}$$

where $\boldsymbol{h}_k$ is the $k$-th optical kernel, $w_k$ is the weight of $\boldsymbol{h}_k$, and $K$ is the total kernel number.

Then a resist model is applied to the aerial image. In our work a constant threshold resist model is used, which sets an intensity threshold $I_{th}$ to binarize the aerial image,

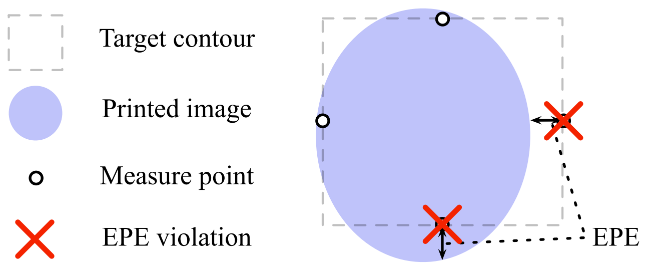


Figure 5.5: Illustration of EPE measurement.

denoted by $\boldsymbol{Z}$ in the following equation.

$$\boldsymbol{Z}(x,y) = f_{resist}(\boldsymbol{I}) = \begin{cases} 1, & \text{if } \boldsymbol{I}(x,y) \geq I_{th}, \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

Finally, binary images $\boldsymbol{Z}_1 = f_{resist}(\boldsymbol{I}_1)$ and $\boldsymbol{Z}_2 = f_{resist}(\boldsymbol{I}_2)$ are combined to form the printed image. Considering that the printed image is binary as well, the process can be represented by performing logical `OR` operation as follows:

$$\boldsymbol{Z}(x,y) = \boldsymbol{Z}_1(x,y) \vee \boldsymbol{Z}_2(x,y). \tag{5.3}$$

### 5.1.2 Evaluation Metrics

Given a target layout and the printed image, the edge placement error (EPE) and process variation band (PV Band) are defined as follows.

**Definition 3 (EPE)** *EPE is defined as the geometric displacement of the image contour from the edge of target image on the layout. A violation is introduced if the perpendicular displacement is greater than an EPE threshold value, as shown in Figure 5.5.*

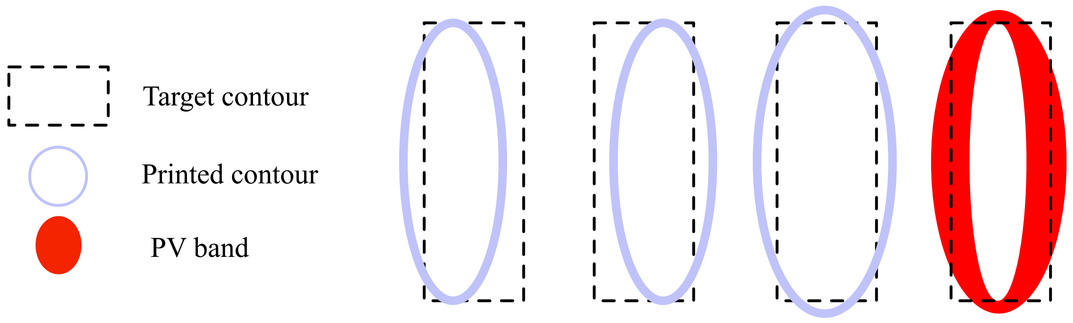


Figure 5.6: Example of PV band.

**Definition 4 (PV Band)** *PV Band is measured as the area between the outermost printed edge and the innermost printed edge among all process conditions, as shown in Figure 5.6. It is used to evaluate the mask robustness against process variations.*

In our implementation, the EPE threshold value is set to $10nm$. An example of our EPE measurement is given in Fig. 5.5: to facilitate the computation, a set of measure points are sampled on each edge and EPE violation will be checked at the measure points. For PV Band, we use XOR operations to compute the region among all possible printed images.

## 5.2 Methodologies

Given the above notations, the problem of layout decomposition and mask optimization (LDMO) is defined as follows.

**Problem 2** (LDMO) *Given target image $\boldsymbol{Z}_t$, two optimized masks, $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$, are generated. The objective is to minimize the difference between the final printed image $\boldsymbol{Z}$ and the target image $\boldsymbol{Z}_t$.*

### 5.2.1 Mathematical Formulation for LDMO

Since we are seeking a pair of masks which can form printed image with high fidelity, the LDMO problem can be formulated as an optimization problem as follows.

$$
\begin{aligned}
\min_{\boldsymbol{M}_1,\boldsymbol{M}_2} \quad & F = \|\boldsymbol{Z}_t - \boldsymbol{Z}\|_2^2 && \text{(5.4a)}\\
\text{s.t.} \quad & \boldsymbol{M}_1(x,y) \in \{0,1\}, \quad \forall x,y, && \text{(5.4b)}\\
& \boldsymbol{M}_2(x,y) \in \{0,1\}, \quad \forall x,y, && \text{(5.4c)}\\
& \boldsymbol{I}_1 = \sum_{k=1}^{K} w_k \cdot |\boldsymbol{M}_1 \otimes \boldsymbol{h}_k|^2, && \text{(5.4d)}\\
& \boldsymbol{I}_2 = \sum_{k=1}^{K} w_k \cdot |\boldsymbol{M}_2 \otimes \boldsymbol{h}_k|^2, && \text{(5.4e)}\\
& \boldsymbol{Z} = f_{resist}(\boldsymbol{I}_1) \vee f_{resist}(\boldsymbol{I}_2). && \text{(5.4f)}
\end{aligned}
$$

The problem is strongly non-convex with discrete constraints, thus is hard to be solved directly. In this section, we propose a unified flow for solving the LDMO problem.

### 5.2.2 Numerical Optimization

Gradient-based method has been widely adopted in solving numerical optimization problems. However, there are non-differentiable discrete constraints in our formulation. Therefore, it is necessary to do relaxation before deriving the gradient.

In the formulation of LDMO, the variables $\boldsymbol{M}_1$, $\boldsymbol{M}_2$, $\boldsymbol{Z}_1$ and $\boldsymbol{Z}_2$ are binary, which are non-differentiable. One possible method is to relax them into floating values with a feasible region of [0,1], which cannot be solved by gradient-based method directly. Alternatively, the binary constraints can be replaced with *sigmoid* function for relaxation so that the variables become unconstrained, and hence convenient to derive the gradient. To apply *sigmoid* function, we need to introduce new variables $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$. Then $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ can be relaxed by applying *sigmoid* function on $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$, respectively (see Equations (5.5) and (5.6)).

$$\boldsymbol{M}_1(x,y) = \mathrm{sig}(\boldsymbol{P}_1(x,y)) = \frac{1}{1+\exp[-\theta_M \boldsymbol{P}_1(x,y)]}, \tag{5.5}$$

$$\boldsymbol{M}_2(x,y) = \mathrm{sig}(\boldsymbol{P}_2(x,y)) = \frac{1}{1+\exp[-\theta_M \boldsymbol{P}_2(x,y)]}. \tag{5.6}$$

Similarly, the *sigmoid* function can be applied to $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$ to relax $\boldsymbol{Z}_1$ and $\boldsymbol{Z}_2$ as shown in Equations (5.7) and (5.8). $\theta_M$ and $\theta_Z$ are user-defined parameters which represent the steepness of *sigmoid* functions, and $I_{th}$ is the threshold in the resist model.

$$\boldsymbol{Z}_1(x,y) = \mathrm{sig}(\boldsymbol{I}_1(x,y)) = \frac{1}{1+\exp[-\theta_Z(\boldsymbol{I}_1(x,y) - I_{th})]}, \tag{5.7}$$

$$\boldsymbol{Z}_2(x,y) = \mathrm{sig}(\boldsymbol{I}_2(x,y)) = \frac{1}{1+\exp[-\theta_Z(\boldsymbol{I}_2(x,y) - I_{th})]}. \tag{5.8}$$

Note that $\boldsymbol{Z}$ is also a binary value, but different from $\boldsymbol{Z}_1$ and $\boldsymbol{Z}_2$, it is calculated by logical OR. We relax constraint (5.4f) to

$$\boldsymbol{Z}(x,y) = \min\{\boldsymbol{Z}_1(x,y) + \boldsymbol{Z}_2(x,y), 1\}. \tag{5.9}$$

Considering that the maximum value of $\boldsymbol{Z}$ before relaxation is 1, here we set an upper bound to 1, which may reduce the error when calculating the objective value. The relation between $\boldsymbol{Z}(x,y)$ and $(\boldsymbol{Z}_1(x,y) + \boldsymbol{Z}_2(x,y))$ is shown in Figure 5.7. Then it is easy to derive the gradient formulation of $\boldsymbol{Z}$ with respect to $\boldsymbol{Z}_1$ and $\boldsymbol{Z}_2$, denoted by

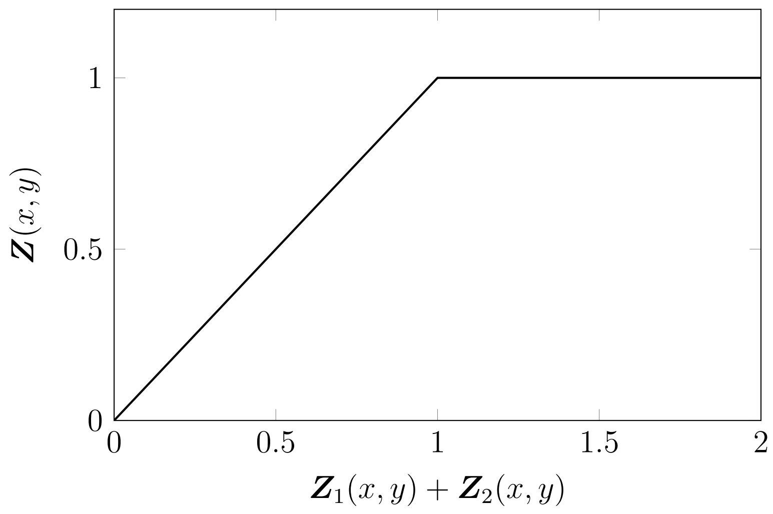


Figure 5.7: Relation between $\boldsymbol{Z}(x,y)$ and $\boldsymbol{Z}_1(x,y)+\boldsymbol{Z}_2(x,y)$.

$\boldsymbol{B}$, which is given by

$$\frac{\partial \boldsymbol{Z}(x,y)}{\partial \boldsymbol{Z}_1(x,y)} = \frac{\partial \boldsymbol{Z}(x,y)}{\partial \boldsymbol{Z}_2(x,y)} = \boldsymbol{B}(x,y) = \begin{cases} 1, & \text{if } \ \boldsymbol{Z}(x,y) \le 1, \\ 0, & \text{otherwise.} \end{cases} \tag{5.10}$$

After relaxation, we can formulate the relaxed LDMO problem as follows.

$$\begin{aligned} \min_{\boldsymbol{P}_1,\boldsymbol{P}_2} \quad & F = \|\boldsymbol{Z}_t - \boldsymbol{Z}\|_2^2 \\ \text{s.t.} \quad & (5.4\text{d}) - (5.4\text{e}), (5.5) - (5.9). \end{aligned} \tag{5.11}$$

Now variables $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ are unconstrained, and functions in Equation (5.5)–

Equation (5.9) are differentiable. We obtain the gradient according to the chain rule.

$$\begin{aligned}\frac{\partial F}{\partial \boldsymbol{P}_1(x,y)} &= \frac{\partial \sum_{i,j}(\boldsymbol{Z}_t(i,j)-\boldsymbol{Z}(i,j))^2}{\partial \boldsymbol{P}_1(x,y)} \\ &= 2\sum_{i,j}(\boldsymbol{Z}(i,j)-\boldsymbol{Z}_t(i,j))\cdot\frac{\partial \boldsymbol{Z}(i,j)}{\partial \boldsymbol{Z}_1(i,j)}\cdot\frac{\partial \boldsymbol{Z}_1(i,j)}{\partial \boldsymbol{P}_1(x,y)}, \qquad (5.12)\end{aligned}$$

where

$$\begin{aligned}\frac{\partial \boldsymbol{Z}_1(i,j)}{\partial \boldsymbol{P}_1(x,y)} &= \theta_M\theta_Z\boldsymbol{Z}(i,j)(1-\boldsymbol{Z}(i,j)) \\ &\quad\times\{[\boldsymbol{M}_1(i,j)\otimes\boldsymbol{H}^*(i,j)]\boldsymbol{H}(i-x,j-y) \\ &\quad\quad+[\boldsymbol{M}_1(i,j)\otimes\boldsymbol{H}(i,j)]\boldsymbol{H}^*(i-x,j-y)\} \\ &\quad\times\boldsymbol{M}_1(i,j)[1-\boldsymbol{M}_1(i,j)]. \qquad (5.13)\end{aligned}$$

Then we can compute the gradient of $F$ with respect to $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ as follows.

$$\begin{aligned}\nabla_{\boldsymbol{P}_1}F &= 2\theta_M\theta_Z\times\boldsymbol{M}_1\odot(1-\boldsymbol{M}_1)\odot \\ &\quad\{\boldsymbol{H}\otimes[(\boldsymbol{Z}-\boldsymbol{Z}_t)\odot\boldsymbol{B}\odot\boldsymbol{Z}\odot(1-\boldsymbol{Z})\odot(\boldsymbol{M}_1\otimes\boldsymbol{H}^*)]+ \\ &\quad\boldsymbol{H}^*\otimes[(\boldsymbol{Z}-\boldsymbol{Z}_t)\odot\boldsymbol{B}\odot\boldsymbol{Z}\odot(1-\boldsymbol{Z})\odot(\boldsymbol{M}_1\otimes\boldsymbol{H})]\}, \qquad (5.14) \\ \nabla_{\boldsymbol{P}_2}F &= 2\theta_M\theta_Z\times\boldsymbol{M}_2\odot(1-\boldsymbol{M}_2)\odot \\ &\quad\{\boldsymbol{H}\otimes[(\boldsymbol{Z}-\boldsymbol{Z}_t)\odot\boldsymbol{B}\odot\boldsymbol{Z}\odot(1-\boldsymbol{Z})\odot(\boldsymbol{M}_2\otimes\boldsymbol{H}^*)]+ \\ &\quad\boldsymbol{H}^*\otimes[(\boldsymbol{Z}-\boldsymbol{Z}_t)\odot\boldsymbol{B}\odot\boldsymbol{Z}\odot(1-\boldsymbol{Z})\odot(\boldsymbol{M}_2\otimes\boldsymbol{H})]\}. \qquad (5.15)\end{aligned}$$

The numerical optimization algorithm is described in Algorithm 3. First we initialize $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$, the maximum iteration number $T$ and the tolerance $\epsilon$ (line 1). An intuitive

**Algorithm 3** Numerical Optimization Flow

```
1: Initialize $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, maximum iteration $T$, tolerance $\epsilon$;
2: for $i = 1, \cdots, T$ do
3:     Compute the printed image according to current $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$;
4:     if printed image is illegal then
5:         Discrete optimization ;                                  ▷ Section 5.2.5
6:     else
7:         MaskUpdate($\boldsymbol{P}_1$, $\boldsymbol{P}_2$);                                ▷ Algorithm 4
8:         if $\text{RMS}(\nabla_{\boldsymbol{P}_1} F) + \text{RMS}(\nabla_{\boldsymbol{P}_2} F) \leq \epsilon$ then
9:             break;
10:        end if
11:    end if
12: end for
```

**Algorithm 4** Gradient-Based Mask Update

```
1: function MaskUpdate($\boldsymbol{P}_1$, $\boldsymbol{P}_2$)
2:     Initialize stepsize $t$;
3:     Compute the relaxed masks $\boldsymbol{M}_1, \boldsymbol{M}_2$;
4:     Compute $\boldsymbol{Z}$ according to current $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$;
5:     Compute the gradient $\nabla_{\boldsymbol{P}_1} F$, $\nabla_{\boldsymbol{P}_2} F$ through Equations (5.4d)–(5.4e), (5.17)–
       (5.18);
6:     $\boldsymbol{P}_1 \leftarrow \boldsymbol{P}_1 - t \times \nabla_{\boldsymbol{P}_1} F$;
7:     $\boldsymbol{P}_2 \leftarrow \boldsymbol{P}_2 - t \times \nabla_{\boldsymbol{P}_2} F$;
8:     return $\boldsymbol{P}_1$,$\boldsymbol{P}_2$, $\nabla_{\boldsymbol{P}_1} F$, $\nabla_{\boldsymbol{P}_2} F$;
9: end function
```

initial solution is that $\boldsymbol{P}_1$ is initialized so that the corresponding $\boldsymbol{M}_1$ is identical to the target image, and $\boldsymbol{P}_2$ is initialized so that $\boldsymbol{M}_2$ is empty. In each iteration, the printed image is obtained based on variables $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ (line 3). The violation checking will be carried out in every iterations, which will be introduced in Section 5.2.4. If the printed image is illegal, a discrete optimization step will be executed, which will be introduced in Section 5.2.5. Otherwise, the function `MaskUpdate` will be called to update the masks. To save the runtime, the loop will exit early if the sum of the root mean square (RMS) of the gradient is less than a tolerance $\epsilon$, which indicates that the objective function may be very close to the optimal value. The loop will finally terminate when the maximum number of iteration $T$ is achieved. In our implementation, $T$ is set to 40. $\theta_M$ and $\theta_Z$ are set to 85 and 4, respectively.

The procedure for mask update is described in Algorithm 4. To derive the gradient, the lithography simulation is conducted first to compute the corresponding printed image based on current masks (line 4). Next, gradient of objective $F$ with respect to $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ are computed (line 5), followed by variables update (lines 6–7). In our implementation, the tolerance $\epsilon$ is set to 0.01 and stepsize $t$ is set to 0.4.

### 5.2.3 Process Variation-aware Mask Optimization

Process variation is a critical issue in realistic manufacturing stage since it can impact the yield directly. Therefore, not only image fidelity but also mask robustness should be taken into considerations. In this section, we extend the LDMO problem to consider process variation issue. Based on the above notations, the problem of process variation-aware layout decomposition and mask optimization is defined as follows.

**Problem 3** (PV-LDMO) *Given target image $\boldsymbol{Z}_t$, two optimized masks, $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$, are generated. The objectives are two fold, (i) minimize the difference between the final printed image $\boldsymbol{Z}$ and the target image $\boldsymbol{Z}_t$, and (ii) the obtained masks, $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$, are robust to process variation.*

PV Band is a commonly applied criterion to evaluate the mask robustness, which can be naturally integrated into the proposed unified optimization framework. The illustration of PV Band is depicted in Section 5.1. It can be observed that the calculation of PV Band requires a series of boolean operations among printed images under all possible process conditions. In order to make the calculation more tractable, we resort to minimize the difference between possible images and the target image, which is in accordance with the target of process variation optimization. The conventional OPC scenario optimizes single mask, therefore the possibility of printed image is the same as the number of process corners $N_p$. Differently, PV-LDMO considers dual masks, thus the possibility of printed image is more than $N_p$ due to the possible combination on separate mask. The PV-LDMO problem can be formulated as below.

$$\min_{\boldsymbol{M}_1,\boldsymbol{M}_2} F_{pv} = \|\boldsymbol{Z}_t - \boldsymbol{Z}\|_2^2 + \sum_{l=1}^{N_p^2} \|\boldsymbol{Z}_t - \boldsymbol{Z}_l\|_2^2 \tag{5.16a}$$

$$\text{s.t.}\ \ \boldsymbol{I}_1^n = \sum_{k=1}^{K} w_{nk} \cdot |\boldsymbol{M}_1 \otimes \boldsymbol{h}_{nk}|^2, n = 1 \ldots N_p, \tag{5.16b}$$

$$\boldsymbol{I}_2^n = \sum_{k=1}^{K} w_{nk} \cdot |\boldsymbol{M}_2 \otimes \boldsymbol{h}_{nk}|^2, n = 1 \ldots N_p, \tag{5.16c}$$

$$\boldsymbol{Z}_l = f_{resist}(\boldsymbol{I}_1^i) \vee f_{resist}(\boldsymbol{I}_2^j),$$

$$i, j = 1 \ldots N_p, l = 1 \ldots N_p^2 \tag{5.16d}$$

$$(5.4b) - (5.4f), \tag{5.16e}$$

where $N_p$ is the number of process windows under consideration. $\boldsymbol{I}_{n1}$ and $\boldsymbol{I}_{n2}$ are the aerial images under the $n$-th process window from $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$, respectively. $\boldsymbol{Z}_l$ is the $l$-th possible combined printed image. Similar to solving Formulation 5.4, the same relaxation scheme can be applied here. Then we can derive the gradient based on the extended objective function and the chain rule as follows.

$$
\begin{aligned}
&\nabla_{\boldsymbol{P}_1} F = 2\theta_M\theta_Z \times \boldsymbol{M}_1 \odot (1-\boldsymbol{M}_1)\odot \\
&\{[\boldsymbol{H} \otimes ((\boldsymbol{Z}-\boldsymbol{Z}_t) \odot \boldsymbol{B} \odot \boldsymbol{Z} \odot (1-\boldsymbol{Z}) \odot (\boldsymbol{M}_1 \otimes \boldsymbol{H}^*))+ \\
&\boldsymbol{H}^* \otimes ((\boldsymbol{Z}-\boldsymbol{Z}_t) \odot \boldsymbol{B} \odot \boldsymbol{Z} \odot (1-\boldsymbol{Z}) \odot (\boldsymbol{M}_1 \otimes \boldsymbol{H}))]+ \\
&\sum_{l=1}^{N_p^2}[\boldsymbol{H} \otimes ((\boldsymbol{Z}_l-\boldsymbol{Z}_t) \odot \boldsymbol{B} \odot \boldsymbol{Z}_l \odot (1-\boldsymbol{Z}_l) \odot (\boldsymbol{M}_1 \otimes \boldsymbol{H}^*))+ \\
&\boldsymbol{H}^* \otimes ((\boldsymbol{Z}_l-\boldsymbol{Z}_t) \odot \boldsymbol{B} \odot \boldsymbol{Z}_l \odot (1-\boldsymbol{Z}_l) \odot (\boldsymbol{M}_1 \otimes \boldsymbol{H}))]\}. \qquad (5.17) \\
&\nabla_{\boldsymbol{P}_2} F = 2\theta_M\theta_Z \times \boldsymbol{M}_2 \odot (1-\boldsymbol{M}_2)\odot \\
&\{\boldsymbol{H} \otimes [(\boldsymbol{Z}-\boldsymbol{Z}_t) \odot \boldsymbol{B} \odot \boldsymbol{Z} \odot (1-\boldsymbol{Z}) \odot (\boldsymbol{M}_2 \otimes \boldsymbol{H}^*)]+ \\
&\boldsymbol{H}^* \otimes [(\boldsymbol{Z}-\boldsymbol{Z}_t) \odot \boldsymbol{B} \odot \boldsymbol{Z} \odot (1-\boldsymbol{Z}) \odot (\boldsymbol{M}_2 \otimes \boldsymbol{H})]\} \\
&\sum_{l=1}^{N_p^2}[\boldsymbol{H} \otimes ((\boldsymbol{Z}_l-\boldsymbol{Z}_t) \odot \boldsymbol{B} \odot \boldsymbol{Z}_l \odot (1-\boldsymbol{Z}_l) \odot (\boldsymbol{M}_2 \otimes \boldsymbol{H}^*))+ \\
&\boldsymbol{H}^* \otimes ((\boldsymbol{Z}_l-\boldsymbol{Z}_t) \odot \boldsymbol{B} \odot \boldsymbol{Z}_l \odot (1-\boldsymbol{Z}_l) \odot (\boldsymbol{M}_2 \otimes \boldsymbol{H}))]\}. \qquad (5.18)
\end{aligned}
$$

Then we follow the numerical optimization algorithm presented in Algorithm 3 to solve the extended PV-LDMO problem with new gradient calculation steps.

### 5.2.4 Violation Detection

If two features of the same mask are too close to each other, there will be a violation in the printed image after lithography simulation and it is difficult to be legalized only through gradient-based optimization. Intuitively, the violation can be resolved if the violated patterns are assigned to different masks. However, we need to locate where the violations occur. To do so, a Hanan-like grid is built based on the geometry of

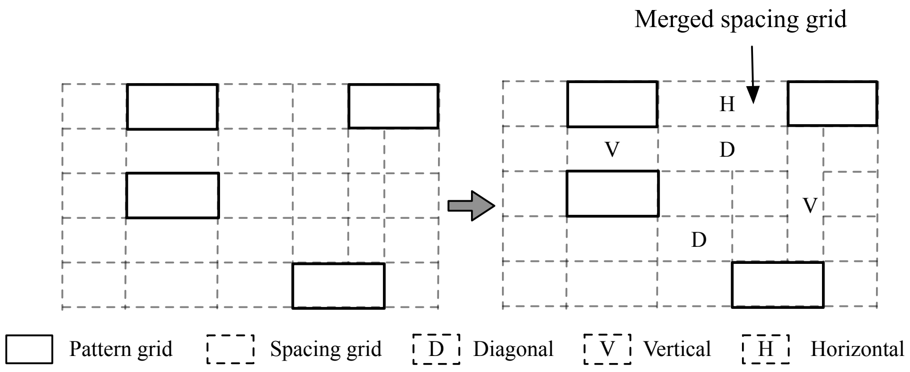


Figure 5.8: Pattern grids and spacing grids.

the bounding box of each target image. Each bounding box shares the same centroid with the pattern inside. Considering that the further assignment is also based on the grid and extra pattern may be generated around the original pattern during the mask optimization, each bounding box is set to be a bit larger than the pattern. In our implementation, the extra width and extra height are both set to 20$nm$. All the grids are then categorized into *pattern grid* and *spacing grid* depending on their positions on the target image. Different from conventional Hanan grid in which all the grids are aligned horizontally and vertically, the adjacent pattern grids in our Hanan-like grid will be merged so that a single pattern will not be split by grids. In addition, the spacing grids between two patterns are also merged into one grid. The orientation of each merged spacing grid is set according to its relative position to the two pattern grids, as shown in Figure 5.8. The H, V and D in Figure 5.8 represent the orientation of horizontal, vertical and diagonal, respectively.

As mentioned before, the violation checking is conducted by every $w$ iterations rather than by each iteration such that the efficiency of the whole flow is maintained. The violation detection is performed through the following way. The printed image is first mapped to the Hanan-like grids. Since all the violations happen at the region

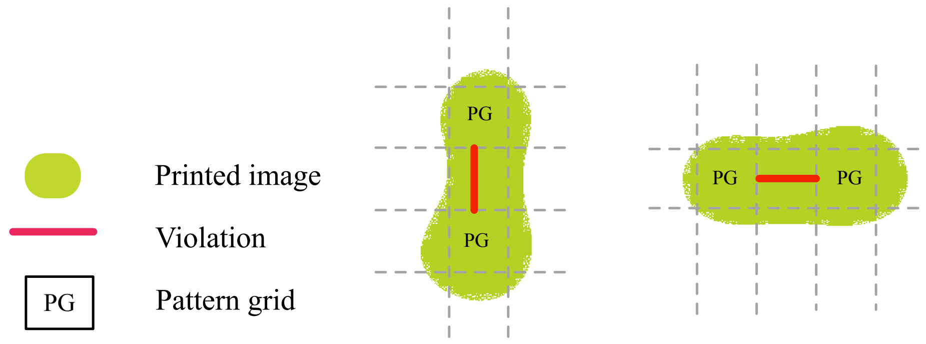


Figure 5.9: Different kinds of violation.

between patterns, i.e., the spacing grids, we only check the spacing grids between two patterns (see Figure 5.9). For each spacing grid, we initialize a matrix $\boldsymbol{A}$ with the same size as the spacing grid, so that each pixel in the grid corresponds to an entry of $\boldsymbol{A}$. Then we find all printable pixels in the grid and set the corresponding entries of the matrix to 1 and set the rest of the entries to 0. An intuitive observation is that a horizontal or vertical violation is caused by a printable line in spacing grids which connects an edge to the opposite side. Diagonal violation is due to the printable lines diagonally connecting two corners. For vertical or horizontal violation, we can check the sum of each row and each column of the matrix $\boldsymbol{A}$. Combined with the orientation, the violation can be determined. If the sum of one row is equal to the width $W$ or the sum of one column is equal to the height $H$, there exists a violation. For diagonal violation, we compute the diagonal length of diagonal spacing grid. If the diagonal length is less than a spacing threshold, there is a diagonal violation. In our implementation, the threshold is set as $110nm$.

### 5.2.5 Discrete Optimization

The masks $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are updated in each iteration to reduce the objective value. Since both LDMO and PV-LDMO are highly non-convex problems, the gradient-based method can only achieve local optimum with poor quality. One reason is that the gradient-based method actually performs a greedy search and it is hard to escape from local optimum. To tackle this problem, we further propose a discrete optimization method to collaborate with the numerical optimization flow.

The most critical issue in multiple patterning lithography is the violations in printed images. Therefore, a critical step of the proposed framework is to resolve violations in the printed image and obtain high quality and robust masks. Generally, resolving violations can be achieved by assigning violated pattern grids to different masks. However, since the correlation between EPE violation and the distribution of patterns is unknown, it is difficult to derive a mathematical formulation to bridge the gap.

To overcome this issue, in this work we develop a discrete optimization approach seeking a two-way partitioning of the pattern grids considering image violation, EPE violation as well as potential spacing rules. Note that the proposed approach here can be easily extended to handle triple patterning lithography, where a three-way partitioning is adopted. With the position of printed image violation and the position of EPE violation, a weighted graph $G(V, E)$ can be constructed, where the vertex $v_i$ represents $i$-th pattern grid and the edges with weight 1 connecting two vertices are conflict with each other. In addition, we add edges with weight $\beta$ between the vertices which have large EPE, where $0 < \beta < 1$. Therefore, the objective of discrete optimization is to find a cut of the graph so that total weight of the edges between the cut and its complement is maximized. We use a vector $\boldsymbol{x}$ to denote the assignment of pattern grids, where

$x_i = 1$ means $v_i$ is assigned to mask 1 and $x_i = -1$ means $v_i$ is assigned to mask 2. Moreover, in order to further ease the mask optimization process, the graph is refined with one more kind of edge. Besides the violation edge and EPE edge, a set of *spacing edges* are added to the graph, which are similar to the diagonal violation edges in the sense that their existence is determined by the distance between each pair of nodes. The threshold for determining a spacing edge is $130nm$. Then the two-way partitioning problem can be formulated as follows.

$$\max_{\boldsymbol{x}} \sum_{(i,j)\in E} w_{ij}(1 - x_i x_j) \tag{5.19a}$$

$$\text{s.t. } x_i \in \{-1, 1\}, \quad \forall v_i \in V, \tag{5.19b}$$

where $w_{ij}$ defines the edge weight as follows.

$$w_{ij} = \begin{cases} 1, & \text{if } v_i \text{ and } v_j \text{ have violation,} \\ \beta, & \text{if } v_i \text{ and } v_j \text{ both have large EPE,} \\ \gamma, & \text{if } v_i \text{ and } v_j \text{ are close but not conflicted,} \\ 0, & \text{otherwise.} \end{cases} \tag{5.20}$$

In our implementation, if the sum of EPE violations of two grids is greater than seven and they are not violated patterns, they will be connected by an edge of weight $\beta$, and $\beta$ is set as 0.1.

Formulation in (5.19) can be approximated to a semidefinite programming (SDP) with Equation (5.21), which can be solved efficiently while maintaining high accuracy.

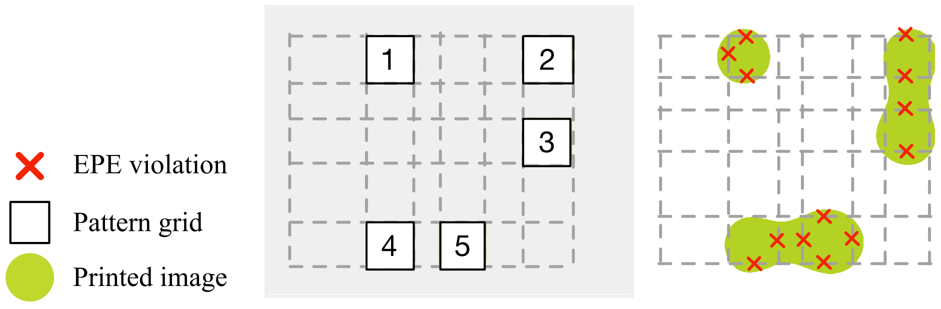
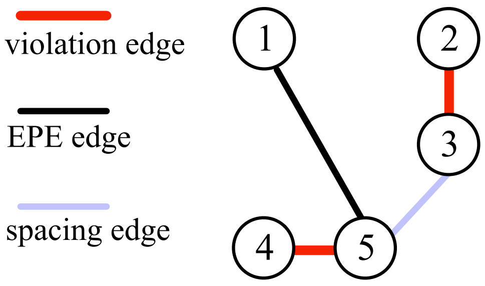


Figure 5.10: (a) Pattern on the mask; (b) Corresponding lithography printed image with EPE violations.



$$\boldsymbol{W} = \begin{bmatrix} 0 & 0 & 0 & 0 & \beta \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & \gamma \\ 0 & 0 & 0 & 0 & 1 \\ \beta & 0 & \gamma & 1 & 0 \end{bmatrix}$$

Figure 5.11: The conflict graph and the corresponding weighted matrix $\boldsymbol{W}$.

$$\min_{\boldsymbol{X}} \quad \boldsymbol{W} \bullet \boldsymbol{X} \tag{5.21a}$$

$$\text{s.t.} \quad \text{diag}(\boldsymbol{X}) = \boldsymbol{e}, \tag{5.21b}$$

$$\boldsymbol{X} \succeq \boldsymbol{0}, \tag{5.21c}$$

where $\boldsymbol{e} = [1, 1 \ldots, 1]^\top$.

The optimal solution $\boldsymbol{X}^*$ of Problem (5.21) need not to be in the form of $\boldsymbol{x}\boldsymbol{x}^\top$, and hence it does not yield a feasible solution to Problem (5.19) immediately. However, we can extract from $\boldsymbol{X}^*$ a solution via randomized rounding [136]. First, we compute Cholesky factorization $\boldsymbol{X}^* = \boldsymbol{U}^\top \boldsymbol{U}$ of $\boldsymbol{X}^*$. The $i$-th column of $\boldsymbol{U}$, denoted by $\boldsymbol{u}_i$, corresponds to the assignment of grid $i$. Let $\boldsymbol{r}$ be a vector uniformly distributed over the unit sphere (i.e., $\|\boldsymbol{r}\|_2 = 1$). Then we can set $x_i$ as follows.

$$x_i = \text{sgn}(\boldsymbol{u}_i^\top \boldsymbol{r}) = \begin{cases} 1, & \text{if } \boldsymbol{u}_i^\top \boldsymbol{r} \geq 0, \\ -1, & \text{otherwise.} \end{cases} \tag{5.22}$$

In other words, we partition the grids according to whether their corresponding vectors lie "above" or "below" the hyperplane. The grids are therefore assigned to different masks according to the value of $x_i$.

An example of graph construction is given in Fig. 5.10 and Fig. 5.11. After solving the SDP, the solution $\boldsymbol{X}^*$ and $\boldsymbol{U}$ are given by

$$\boldsymbol{X}^* = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & -1 \\ -1 & 0 & 0 & -1 & 1 \end{bmatrix}, \boldsymbol{U} = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

With a random vector $\boldsymbol{r} = [r_1, r_2 \cdots r_5]^\top$ and Equation (5.22), vector $\boldsymbol{x}$ is obtained by

$$\boldsymbol{x} = [\text{sgn}(r_1), \text{sgn}(r_2), \text{sgn}(-r_2), \text{sgn}(r_1), \text{sgn}(-r_1)]^\top. \tag{5.23}$$

By solving SDP we can obtain multiple solutions which are useful to avoid being stuck in local optimum during the succeeding numerical optimization process. Furthermore, the runtime cost of solving SDP is much smaller than lithography simulation.

Therefore, SDP is an efficient method for our discrete optimization. Once the SDP is solved, we can obtain multiple solutions for $\boldsymbol{x}$. Then all these obtained solutions are further optimized through numerical optimization flow without violation checking, where these solutions will go through a pruning process and sub-optimal ones will be removed.

The detailed procedure is shown in Algorithm 5. Firstly, an empty set P is initialized to store the potential solutions (line 1). Then $S$ solutions are obtained by randomized rounding, from which we can get the corresponding assignment solution for two masks, i.e., $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$. Each pair of $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ is treated as a 2-tuple which is stored in P (lines 2–6). Next, gradient-based mask update illustrated in Algorithm 3 is performed for $T$ iterations for each solution in P. After that, the legality of the printed image of each solution will be checked. The solutions generating illegal printed images will be discarded first. Then the number of EPE violations of all the solutions will be compared and half of the solutions with larger EPE violation will be discarded (lines 8–15). The pruning process will be repeated until only one element in P is left. In order to balance the runtime and performance, $T$ and $S$ are both set to 5 in our implementation.

Compared with the graph construction method in [7], our conflict graph construction can reflect the lithography principles better by considering the spacing rules. The advantage is two fold. (1) By adding the spacing edges with weight $\gamma$, the solution space of the SDP is tighter than that in [7], which helps the search procedure to focus more on the superior solutions. (2) The runtime of the search and pruning routine (Algorithm 5) depends on the number of the solutions we obtained from the SDP solver. Constraining the solutions space in a superior region also accelerates the algorithm to converge. These advantages can be verified in our experimental results.

**Algorithm 5** Pruning

```
Require: SDP solution $\boldsymbol{X}^*$.
Ensure: $\boldsymbol{P}_1$, $\boldsymbol{P}_2$.
1: Initialize set P;
2: for $i \leftarrow 1, \cdots, S$ do
3:     Randomized rounding;                                  ▷ Equation (5.22)
4:     Get corresponding $\boldsymbol{P}_1^i$, $\boldsymbol{P}_2^i$;          ▷ $\{\boldsymbol{P}_1, \boldsymbol{P}_2\}$ is a 2-tuple
5:     Save $\{\boldsymbol{P}_1^i, \boldsymbol{P}_2^i\}$ in P;
6: end for
7: while P.size()$> 1$ do
8:     for all $\{\boldsymbol{P}_1^i, \boldsymbol{P}_2^i\} \in$ P do
9:         for $j \leftarrow 1, \cdots, T$ do
10:            MaskUpdate($\boldsymbol{P}_1^i, \boldsymbol{P}_2^i$);          ▷ Algorithm 4
11:        end for
12:        Check legality of the printed image and get the number of EPE violations;
13:    end for
14:    Remove solutions with illegal printed image;
15:    Remove half of solutions with larger EPE violations;
16: end while
17: return the remaining $\{\boldsymbol{P}_1, \boldsymbol{P}_2\}$;
```

### 5.2.6 Overall Flow

From the problem formulation, it is clear that the LDMO and PV-LDMO are both strongly non-convex problems without analytic structure, which makes them numerically hard to solve. In order to solve the problem, we design a unified optimization flow which includes a numerical optimization flow and a discrete optimization flow. These two engines are collaborative with each other. Basically, the masks are optimized numerically with gradient-based optimization. Once the violation is detected in the printed image, another SDP-based discrete optimization engine is triggered to resolve the violations. Multiple solutions are obtained from the solution of SDP, which can help to jump out of local optimum and act as a guidance of numerical optimization. The solutions returned by SDP will be numerically optimized until a pair of masks with

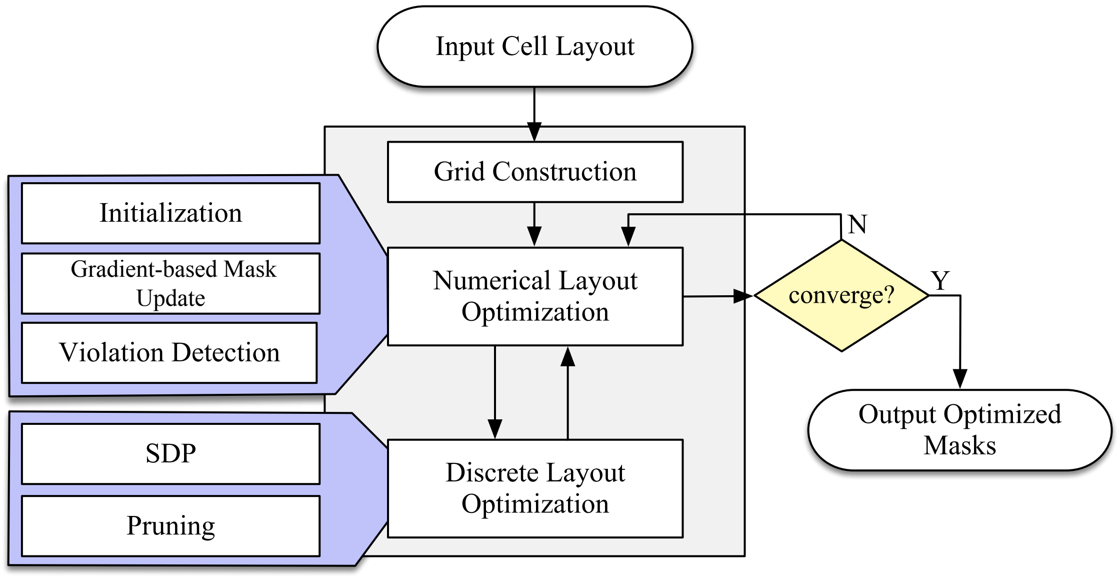


Figure 5.12: Overall LDMO flow.

highest quality is selected, which will be further optimized by numerical optimization flow. The overall flow is presented in Figure 5.12.

## 5.3 Experimental Results

### 5.3.1 Environment and Implementation Details

We implement our algorithms with C++ on an Intel Core 2.6 GHz Linux machine with 48 GB RAM. To solve SDP we use Csdp, a package for specifying and solving semidefinite programming problems [137]. We use an open source lithography simulator and EPE checker [138], where the intensity threshold is set to 0.039. The EPE violation threshold value is set to $10nm$, which is more strict than the $15nm$ used in [138]. The experiments are conducted using NanGate, an open-source standard cell library [139].

Table 5.2: Comparison between previous flow and LDMO. $\text{EPE}_{\text{thre}} = 10nm$.

| Cell | ENUM +[5] | | | | [4] +[5] | | [6] +[5] | | [7] | | Ours | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | #LD | #Complete | #EPEV | RT (s) | #EPEV | RT (s) | #EPEV | RT (s) | #EPEV | RT (s) | #EPEV | RT (s) |
| `INV_X1` | 10 | 10 | 1 | 11814 | 1 | 1183 | 1 | 1183 | 1 | 1995 | 0 | 695 |
| `NOR2_X1` | 18 | 18 | 2 | 25117 | 4 | 1405 | 8 | 1421 | 1 | 1996 | 0 | 642 |
| `BUF_X1` | 22 | 22 | 1 | 23072 | 5 | 1068 | 5 | 867 | 1 | 1990 | 0 | 773 |
| `CLKBUF_X1` | 30 | 30 | 1 | 30486 | 1 | 737 | 1 | 1046 | 0 | 1996 | 0 | 890 |
| `OAI211_X1` | 34 | 34 | 3 | >36000 | 7 | 1207 | 5 | 1213 | 6 | 1996 | 2 | 950 |
| `AOI211_X1` | 42 | 31 | 3 | >36000 | 8 | 1080 | 5 | 1048 | 1 | 1989 | 1 | 937 |
| `AND2_X1` | 52 | 23 | 3 | >36000 | 13 | 1061 | 13 | 1080 | 1 | 1993 | 0 | 720 |
| `OR2_X1` | 82 | 33 | 0 | >36000 | 3 | 1220 | 7 | 1046 | 0 | 1997 | 0 | 760 |
| `NAND4_X1` | 86 | 31 | 3 | >36000 | 6 | 1176 | 5 | 1173 | 1 | 1997 | 0 | 809 |
| `NAND3_X2` | 252 | 35 | 6 | >36000 | 8 | 774 | 7 | 1171 | 3 | 1998 | 2 | 680 |
| `OR4_X1` | 1282 | 33 | 1 | >36000 | 2 | 1233 | 5 | 1188 | 0 | 1987 | 0 | 730 |
| `NOR3_X2` | 3016 | 37 | 3 | >36000 | 4 | 1168 | 5 | 773 | 7 | 1999 | 3 | 924 |
| `OAI33_X1` | 6178 | 40 | 4 | >36000 | 10 | 922 | 11 | 1017 | 6 | 1998 | 0 | 811 |
| Average | | | 2.55 | >34595.35 | 6.09 | 1058.73 | 6.27 | 1056.55 | 2.15 | 1994.62 | 0.46 | 793.92 |
| Ratio | | | 5.54 | >43.34 | 13.23 | 1.45 | 13.63 | 1.42 | 4.67 | 2.51 | **1.00** | **1.00** |

Table 5.3: Comparison between previous flow and LDMO. $\text{EPE}_{\text{thre}} = 8nm$.

| Cell | ENUM +[5] | | | | [4] +[5] | | [6] +[5] | | [7] | | Ours | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | #LD | #Complete | #EPEV | RT (s) | #EPEV | RT (s) | #EPEV | RT (s) | #EPEV | RT (s) | #EPEV | RT (s) |
| INV_X1 | 10 | 10 | 3 | 11865 | 6 | 1201 | 6 | 1201 | 4 | 1994 | 0 | 684 |
| NOR2_X1 | 18 | 18 | 6 | 25217 | 10 | 1496 | 11 | 1436 | 4 | 1993 | 0 | 623 |
| BUF_X1 | 22 | 22 | 4 | 23988 | 12 | 1165 | 9 | 1063 | 4 | 1990 | 0 | 706 |
| CLKBUF_X1 | 30 | 30 | 1 | 30486 | 1 | 878 | 1 | 936 | 3 | 1990 | 1 | 846 |
| OAI211_X1 | 34 | 34 | 3 | >36000 | 7 | 1288 | 5 | 1265 | 7 | 1999 | 5 | 911 |
| AOI211_X1 | 42 | 31 | 3 | >36000 | 13 | 1195 | 10 | 1132 | 6 | 1992 | 4 | 897 |
| AND2_X1 | 52 | 23 | 6 | >36000 | 17 | 1123 | 18 | 1126 | 9 | 1993 | 3 | 708 |
| OR2_X1 | 82 | 33 | 2 | >36000 | 4 | 1309 | 10 | 1160 | 1 | 1992 | 1 | 751 |
| NAND4_X1 | 86 | 31 | 9 | >36000 | 9 | 1256 | 13 | 1221 | 6 | 1999 | 2 | 792 |
| NAND3_X2 | 252 | 35 | 10 | >36000 | 11 | 903 | 15 | 1209 | 6 | 1998 | 2 | 659 |
| OR4_X1 | 1282 | 33 | 6 | >36000 | 7 | 1320 | 9 | 1236 | 1 | 1987 | 1 | 719 |
| NOR3_X2 | 3016 | 37 | 6 | >36000 | 10 | 1197 | 11 | 963 | 9 | 1999 | 4 | 906 |
| OAI33_X1 | 6178 | 40 | 8 | >36000 | 13 | 1010 | 14 | 1123 | 5 | 1998 | 2 | 799 |
| Average | | | 5.15 | >31965.85 | 9.23 | 1180.08 | 10.15 | 1059.31 | 5.00 | 1994.15 | 1.92 | 769.31 |
| Ratio | | | 2.68 | >41.55 | 4.80 | 1.53 | 5.28 | 1.51 | 2.60 | 2.59 | **1.00** | **1.00** |

Table 5.4: Comparison between previous flow and LDMO. $\text{EPE}_{\text{thre}} = 5nm$.

| Cell | ENUM +[5] | | | | [4] +[5] | | [6] +[5] | | [7] | | Ours | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | #LD | #Complete | #EPEV | RT (s) | #EPEV | RT (s) | #EPEV | RT (s) | #EPEV | RT (s) | #EPEV | RT (s) |
| INV_X1 | 10 | 10 | 15 | 12936 | 21 | 1200 | 21 | 1201 | 8 | 1992 | 7 | 674 |
| NOR2_X1 | 18 | 18 | 21 | 26214 | 27 | 1496 | 22 | 1436 | 12 | 1993 | 15 | 652 |
| BUF_X1 | 22 | 22 | 19 | 25632 | 29 | 1153 | 24 | 1100 | 19 | 1994 | 7 | 720 |
| CLKBUF_X1 | 30 | 30 | 18 | 31995 | 23 | 896 | 23 | 953 | 12 | 1991 | 4 | 855 |
| OAI211_X1 | 34 | 34 | 18 | >36000 | 27 | 1263 | 27 | 1250 | 25 | 1999 | 23 | 898 |
| AOI211_X1 | 42 | 31 | 21 | >36000 | 38 | 1185 | 33 | 1136 | 21 | 1999 | 17 | 909 |
| AND2_X1 | 52 | 23 | 25 | >36000 | 30 | 1163 | 33 | 1130 | 21 | 1999 | 12 | 782 |
| OR2_X1 | 82 | 33 | 21 | >36000 | 24 | 1286 | 26 | 1230 | 15 | 1998 | 4 | 753 |
| NAND4_X1 | 86 | 31 | 32 | >36000 | 38 | 1263 | 41 | 1236 | 18 | 1998 | 10 | 799 |
| NAND3_X2 | 252 | 35 | 42 | >36000 | 50 | 889 | 52 | 1213 | 19 | 1999 | 13 | 702 |
| OR4_X1 | 1282 | 33 | 28 | >36000 | 30 | 1213 | 34 | 1250 | 15 | 1998 | 11 | 730 |
| NOR3_X2 | 3016 | 37 | 31 | >36000 | 38 | 1163 | 37 | 956 | 18 | 1997 | 12 | 931 |
| OAI33_X1 | 6178 | 40 | 38 | >36000 | 41 | 1110 | 52 | 1020 | 24 | 1999 | 15 | 821 |
| Average | | | 25.31 | >32367.46 | 32.00 | 1175.38 | 32.69 | 1162.38 | 17.46 | 1994.62 | 11.54 | 786.62 |
| Ratio | | | 2.19 | >41.15 | 2.77 | 1.49 | 2.83 | 1.48 | 1.51 | 2.54 | **1.00** | **1.00** |

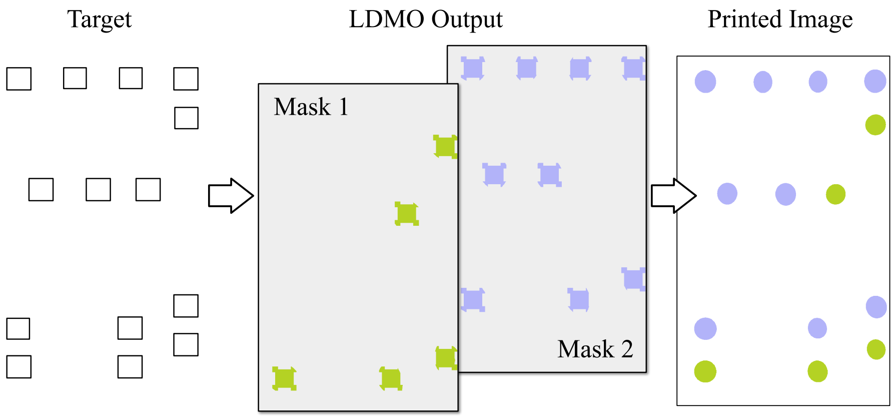


Figure 5.13: Example of our layout decomposition and mask optimization on cell `OR2_X1`.

We test the proposed unified framework on contact layers where stitches are forbidden. Figure 5.13 gives an example of our output masks and the printed images for cell `OR2_X1`. All the solutions obtained are legal in our experiments. We implement a layout decomposition engine, where branch-&-bound methodology is applied to search for all legal coloring solutions. In layout decomposition, the coloring distance is set to 110$nm$, so all contact layers are double patterning friendly. We obtain a modified binary of mask optimization engine from [5].

### 5.3.2 Results of LDMO

In the first experiment, we compare the proposed framework with an exhaustive optimization flow, where all legal layout decomposition solutions are enumerated, and all the solutions are fed into the mask optimization engine [5]. The results of the exhaustive optimization are shown in the merged column "ENUM + [5]" of Table 5.2. The column "#LD" represents the total number of enumerated layout decomposition solutions. Considering that it will take extremely long time if we run mask optimization on all layout decomposition solutions, we set an upper bound of runtime, which is 36000 seconds (i.e., 10 hours). The column "#Complete" lists the total number of solutions that have been finished within the runtime limit. Then we can obtain the best decomposed layout with the least EPE violations. The columns "#EPEV" and "RT (s)" list the best EPE violation number and the total mask optimization runtime in seconds. Note that compared to expensive mask optimization, the runtime of layout decomposition is usually ignorable. The corresponding results of our unified framework are shown in the merged column "Ours". Compared with the exhaustive optimization flow, our unified framework can effectively reduce the EPE violations number by 8×, meanwhile it can achieve more than 35× speed-up on average.

In order to avoid the unrealistic runtime cost of the exhaustive optimization, heuristic selection methods were proposed in previous work by Yu *et al.* [4] and Chen *et al.* [6]. In the second experiment, we use these two strategies to select from the exhaustive layout decomposition solutions, and feed the selected solutions to mask optimization engine of [5]. Then we compare the quality of corresponding printed patterns with ours. The corresponding results are shown in merged columns "[4] +[5]" and "[6] +[5]" in Table 5.2. Here columns "#EPEV" and "RT (s)" represent the EPE violation number and the runtime of mask optimization on the selected layout decomposition solutions. From the table we can see that our proposed framework can achieve around 65% and 66% EPE violation reduction compared to the heuristic selection in [4] and [6], respectively. The experimental results show that the density based layout decomposition strategy may not promise an optimal printed image quality after mask optimization.

Compared with the preliminary work [7], new violation graph is constructed as described in Section 5.2.5, which boosts the performance in terms of both image quality and runtime. On average, with the new techniques, LDMO achieves 4× reduction on the number of EPE violations, and more than 2× speed-up.

In addition, we also have explored the how the proposed framework perform under stricter EPE violation threshold conditions. We further use $8nm$ and $5nm$ as the threshold for determining an EPE violation, and the results are presented in Tables 5.3 and 5.4. It can be observed that the proposed framework can still outperform all the baselines in terms of the quality of the obtained masks and the running time.

For comprehensive comparison among the three flows listed in Table 5.2, we plot the distribution of EPE violations of all enumerated solutions of a cell according to the results of "ENUM+[5]" (see Figure 5.14). We can find that different solutions of layout decomposition result in diverse EPE cost after mask optimization. The solution obtained by different methods are marked in the figure. It can also be seen that among all the potential solutions, most coloring solutions are actually sub-optimal, while methods proposed in [4] and [6] do not select the optimal ones which correspond to the leftmost bar in each chart. Take Figure 5.14a as an example. There are 22 DPLD solutions, while the number of EPE violations of these solutions ranges from 1 to 9, among which half of the solutions have 5 EPE violations, including the solutions selected by methods proposed in [4] and [6]. The masks generated by our method can achieve 0 violation. The same observation can also be found from Figure 5.14c. We can see that the quality of the masks obtained by unified optimization can even outperform all the solutions obtained by conventional two-stage flow.

Figure 5.15 demonstrates the convergence of the EPE violation number. Since the optimization process will not get stuck in local optimum, it can be seen that the number of EPE violations goes up on some iterations. Eventually, it will converge to a solution with fewer EPE violations.

Examples of printed images of the contact layers are given in Figures 5.16 to 5.18 in which the EPE violations are marked with red cross on the pattern. It can be seen more explicitly that the proposed algorithm can find masks with higher quality, which have fewer EPE violations on printed images.

### 5.3.3 Results of PV-LDMO

In practical manufacturing process, the process variation is also a critical issue to be considered. The proposed PV-LDMO flow is designed for this target. Next, we demonstrate the experimental results of the PV-LDMO flow. The statistics are presented in Figures 5.19 to 5.21. In the figures, “NG” and “PV” denote new graph construction and process variation-aware optimization, respectively. We use the fundamental LDMO flow proposed by the preliminary work [7] as baseline. It can be noted that by considering the new PV Band optimization objective and applying new layout representation graph, the number of EPE violations and the PV Band area are reduced significantly. Compared with the preliminary work [7], the number of EPE violation is reduced by 84%, while the PV Band area is improved by 2%. What’s more, the new flow achieves 2× speedup on runtime.

We verify the effectiveness of each new presented technique of the PV-LDMO flow by conducting comprehensive ablation study. To do so, each newly designed optimization technique is enabled separately to justify the benefit of its own. The results can also be observed in Figures 5.19 to 5.21. It is shown that applying only new graph construction will result in the shortest runtime and smaller number of EPE violations. Solely applying PV Band optimization will result in the smallest PV Band area, while the number of EPE violation will increase. Leveraging both of them can take the good side of each and lead to satisfying results on all aspects.

### 5.3.4 Robustness and Pattern Density

In the experiment, we use a straight-forward method for initialization. $\boldsymbol{M}_1$ is initialized with target image $\boldsymbol{Z}$ and $\boldsymbol{M}_2$ is initialized as a 0 matrix (empty mask), which is a conventional initialization approach for mask optimization [5, 66]. The decomposition/assignment happens in the discrete optimization. Different initial assignments may be obtained at the first discrete optimization process due to the randomized rounding and pruning, which may lead to different final solutions. In order to analyze the sensitivity of the initial decomposition to the proposed framework, we conduct multiple runs and record all the results which is shown as in Table 5.5. It can be seen that there are slight vibration among different runs while the overall results is robust to the randomness.

In double patterning process, the pattern density uniformity between two masks is of great importance in manufacturing. The final on-wafer image is generated by etching process which is sensitive to the pattern density. Since the proposed framework is targeting at simultaneous layout decomposition and mask optimization in multiple patterning process, pattern density issue should be taken into consideration. It can be realized in two ways. (1) Pruning solutions based on a combined metric considering both printed image quality and pattern density uniformity (e.g., a weighted summation of these two metrics); (2) Set a hard constraint on the uniformity and directly pruned unsatisfied ones. To verify the idea, we implement a uniformity-aware discrete optimization and conducted the experiments. Since in our experiments the total regions

Table 5.5: Results record of multiple experimental runs

| Cell | 1 | | 2 | | 3 | | 4 | | 5 | | Average | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | EPEV # | PVB ($nm^2$) | EPEV # | PVB ($nm^2$) | EPEV # | PVB ($nm^2$) | EPEV # | PVB ($nm^2$) | EPEV # | PVB ($nm^2$) | EPEV # | PVB ($nm^2$) |
| `INV_X1` | 0 | 14247 | 0 | 13944 | 0 | 13956 | 0 | 14332 | 0 | 14392 | 0.0 | 14174 |
| `NOR2_X1` | 0 | 19278 | 1 | 18667 | 0 | 19278 | 1 | 18802 | 0 | 19289 | 0.4 | 19063 |
| `BUF_X1` | 0 | 19070 | 1 | 19011 | 0 | 18917 | 0 | 19023 | 0 | 19064 | 0.2 | 19017 |
| `CLKBUF_X1` | 0 | 17139 | 0 | 17936 | 0 | 17140 | 0 | 17633 | 0 | 16949 | 0.0 | 17359 |
| `OAI211_X1` | 1 | 31038 | 0 | 31214 | 1 | 31333 | 0 | 31292 | 2 | 31075 | 0.8 | 31190 |
| `AOI211_X1` | 0 | 30455 | 1 | 30920 | 0 | 30478 | 0 | 30386 | 0 | 30936 | 0.2 | 30635 |
| `AND2_X1` | 0 | 23493 | 0 | 23590 | 4 | 22780 | 0 | 23468 | 0 | 23402 | 0.8 | 23347 |
| `OR2_X1` | 0 | 22327 | 0 | 22038 | 0 | 22505 | 0 | 22327 | 1 | 22641 | 0.2 | 22368 |
| `NAND4_X1` | 0 | 29235 | 1 | 28607 | 0 | 28524 | 0 | 28359 | 1 | 28430 | 0.4 | 28667 |
| `NAND3_X2` | 0 | 36531 | 0 | 36753 | 0 | 36766 | 1 | 36255 | 0 | 36226 | 0.2 | 36506 |
| `OR4_X1` | 0 | 28398 | 0 | 28569 | 0 | 28972 | 0 | 28639 | 0 | 28644 | 0.0 | 28644 |
| `NOR3_X2` | 0 | 30969 | 2 | 31457 | 2 | 31272 | 0 | 31435 | 3 | 31574 | 1.4 | 31341 |
| `OAI33_X1` | 0 | 33295 | 0 | 33460 | 0 | 33363 | 0 | 33704 | 1 | 33464 | 0.2 | 33583 |

Table 5.6: Performance of pattern uniformity-aware LDMO

| Cell | NG only | | PV only | | NG+PV | |
|---|---|---|---|---|---|---|
| | EPEV # | PVB ($nm^2$) | EPEV # | PVB ($nm^2$) | EPEV # | PVB ($nm^2$) |
| INV_X1 | 0 | 14189 | 0 | 13946 | 0 | 13951 |
| NOR2_X1 | 0 | 19253 | 0 | 18742 | 0 | 18987 |
| BUF_X1 | 0 | 19142 | 0 | 19055 | 0 | 18996 |
| CLKBUF_X1 | 0 | 17742 | 0 | 17296 | 0 | 16992 |
| OAI211_X1 | 1 | 31354 | 1 | 30897 | 1 | 31244 |
| AOI211_X1 | 1 | 30561 | 2 | 30496 | 0 | 30478 |
| AND2_X1 | 0 | 23861 | 0 | 23869 | 2 | 23798 |
| OR2_X1 | 0 | 22478 | 0 | 22190 | 0 | 22148 |
| NAND4_X1 | 0 | 28429 | 4 | 26795 | 0 | 28210 |
| NAND3_X2 | 1 | 37232 | 1 | 35931 | 0 | 35894 |
| OR4_X1 | 0 | 28478 | 1 | 28173 | 0 | 28403 |
| NOR3_X2 | 3 | 31280 | 2 | 31096 | 2 | 30703 |
| OAI33_X1 | 1 | 33467 | 0 | 33282 | 1 | 33472 |
| Average | 0.54 | 26805 | 0.85 | 25520 | 0.46 | 25636 |
| Ratio | 1.17 | 1.04 | 1.85 | 0.99 | **1.0** | **1.0** |

of the two masks are the same, we use the pattern area on each mask to represent the pattern density. Denote the pattern density on the two masks as $D_1$ and $D_2$, respectively. They can be calculated using $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$, or can be approximately calculated using the solution (Equation (5.23)). The constraint is set to be $|D_1 - D_2| \leq 20\% \times \mathcal{D}$, where $\mathcal{D}$ is the total area of the target patterns. The results are shown in Table 5.6 after applying the uniformity-aware constraint.

## 5.4 Discussion

### 5.4.1 Handling Variable Threshold Resist Model

The relaxation approach applied in Equation (5.7) and Equation (5.8) assumes a constant threshold model (CTM) when forming a printed image from an aerial image.

The proposed Algorithm 3 and Algorithm 4 are also applicable to the scenario where a variable threshold model (VTM) is used. Recall that CTM is relaxed using sigmoid function to incorporate with numerical optimization (Equations (5.7) and (5.8)). Similar approaches can be used if the resist model is VTM. Essentially, we just need to obtain pixel-wise threshold before binarizing the aerial image to printed image. Suppose a VTM is given as $I_{vth} = C_1 + C_2 \times I_{\max}$ [140], where $I_{max}$ is the maximum intensity of a local region of aerial image, $I_{vth}$ is the threshold of the pixels within the same region. $C_1$ and $C_2$ are the parameters in the model. After aerial images $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$ are generated, two threshold matrices $\boldsymbol{I}_{vth1}$ and $\boldsymbol{I}_{vth2}$ can be obtained, which have the same dimension as the corresponding aerial images. The entries in $\boldsymbol{I}_{vth1}$ and $\boldsymbol{I}_{vth2}$ indicate the threshold of the corresponding pixels in $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$, respectively. Then the Equations (5.7) and (5.8) can be written as

$$\boldsymbol{Z}_1(x,y) = \frac{1}{1+\exp[-\theta_Z(\boldsymbol{I}_1(x,y) - \boldsymbol{I}_{vth1}(x,y))]}, \tag{5.24}$$

$$\boldsymbol{Z}_2(x,y) = \frac{1}{1+\exp[-\theta_Z(\boldsymbol{I}_2(x,y) - \boldsymbol{I}_{vth2}(x,y))]}. \tag{5.25}$$

Considering the $\boldsymbol{I}_{vth1}$ and $\boldsymbol{I}_{vth2}$ are only correlated with the maximum local intensity in $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$, thus they can be treated as constants when deriving the gradient based on above equations, which makes the gradient calculation the same as Equations (5.17) and (5.18).

### 5.4.2 Handling More Advanced Multiple Patterning Lithography

The proposed framework contains a gradient-based numerical optimization as well as a set of discrete optimization. The main step for gradient-based numerical optimization is the mathematical relaxation and derivation of the gradient for each individual mask. For triple patterning process, the problem formulation can be rewritten with an additional variable $\boldsymbol{M}_3$ and additional constraints.

$$\boldsymbol{M}_3(x, y) \in \{0, 1\}, \quad \forall x, y, \tag{5.26}$$

$$\boldsymbol{I}_3 = \sum_{k=1}^{K} w_k \cdot |\boldsymbol{M}_3 \otimes \boldsymbol{h}_k|^2, \tag{5.27}$$

$$\boldsymbol{Z} = f_{resist}(\boldsymbol{I}_1) \vee f_{resist}(\boldsymbol{I}_2) \vee f_{resist}(\boldsymbol{I}_3). \tag{5.28}$$

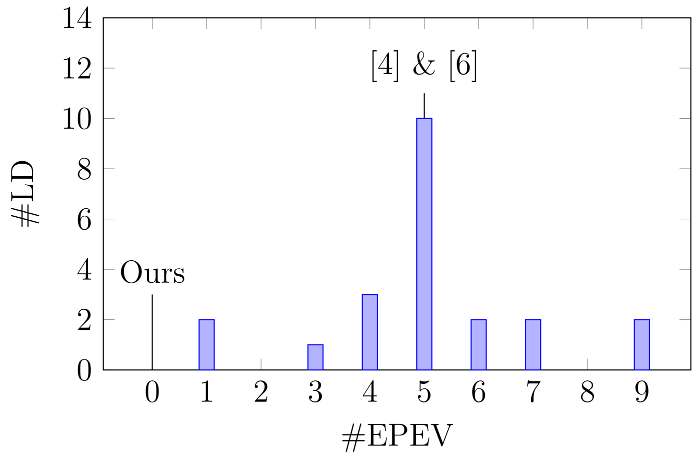
It can be observed that the gradient derivation is not limited to only 2 masks, since all the mathematical relaxation methods can be applied to the newly added variables and constraints, and a similar equation can be derived for $\boldsymbol{M}_3$. The discrete optimization requires a slightly change to handle triple patterning lithography cases which is essentially a three-way partition. For double patterning process, the problem is formulated into a max-cut problem and relaxed to an SDP formulation. For triple patterning process, discrete optimization solutions can be generated based on vector programming which also can be relaxed to an SDP formulation, as proposed in [141, 53]. Considering that there may be some patterns that have multiple assignment choices, we can also obtain multiple solutions from the three-way partition step. Similarly, the pruning step is applied to select the most promising solution containing a 3-tuple, and proceed as shown in Figure 5.12.

### 5.4.3 Handling Other Layers

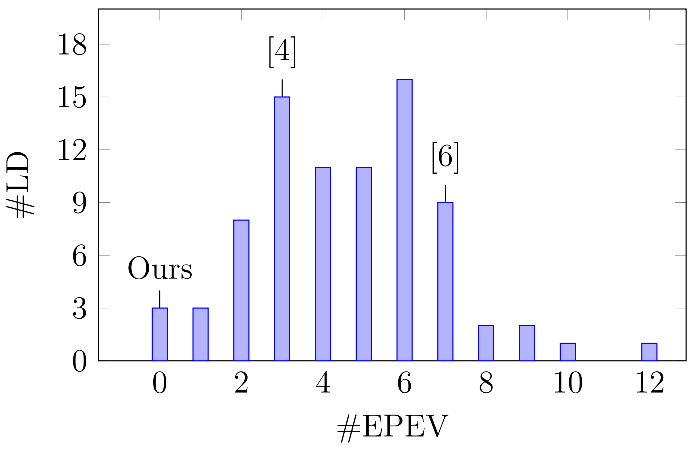
In addition, although the experiments are conducted with contacted layers in this work, the optimization methodologies proposed in this framework are general. Specifically, the gradient-based numerical optimization is a conventional technique which is generally applicable to any layers. The discrete optimization is designed to assist the conflicting pattern separation and help to explore higher quality solutions, therefore the fundamental idea is applicable to other layers. In order to perform LDMO for other layers, there are a few detailed modifications can be applied to some of the sub-modules. Particularly, the grid construction needs to incorporate other issues like stitch insertion, which essentially requires more engineering efforts for stitch insertion [50, 53] and grid merging, such that the violation detection, discrete optimization and pruning can still perform the functionality as they are.

## 5.5 Summary

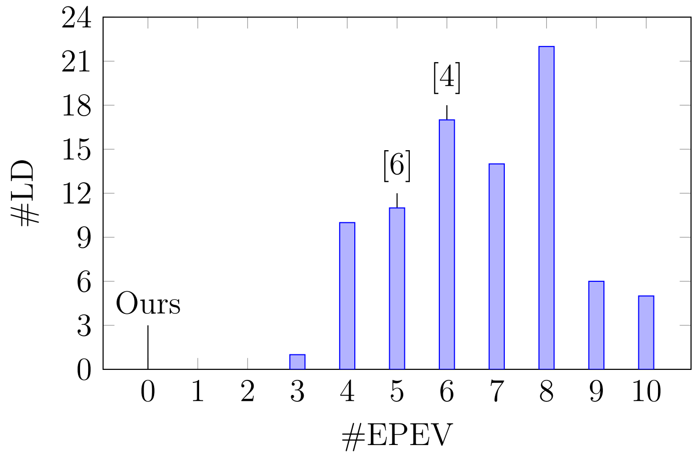
In this chapter we have proposed a unified framework solving layout decomposition and mask optimization problem, while taking process variation issues into consideration. In this framework, we designed two collaborative flows for optimization: a gradient-based numerical optimization, as well as a set of discrete optimizations to jump out of local optimum. The experimental results show that our proposed framework outperforms conventional flow in terms of both runtime and EPE violation number. To the best of our knowledge, this is the first work trying to handle multiple patterning layout decomposition and mask optimization simultaneously. Note that our framework is general and it can be extended to handle triple or quadruple patterning lithography coloring rules. We hope this work can stimulate more future work into this field.



(a)



(b)



(c)

Figure 5.14: Distribution of #EPE violations of different cells: (a) `BUF_X1`; (b) `OR2_X1`; (c) `NAND4_X1`.

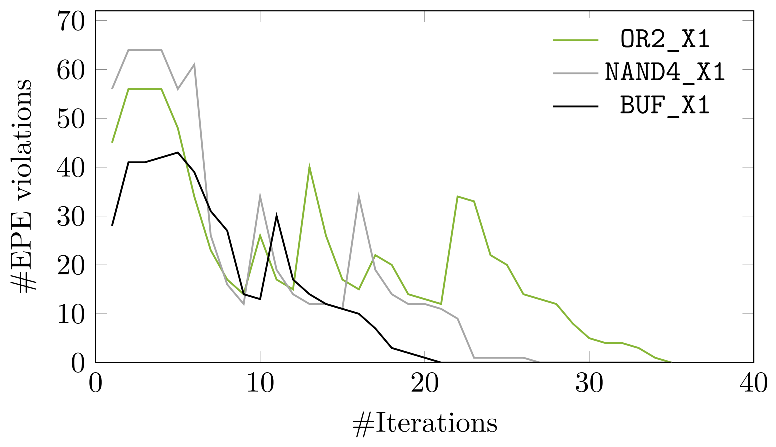


Figure 5.15: Convergence of #EPE violations.

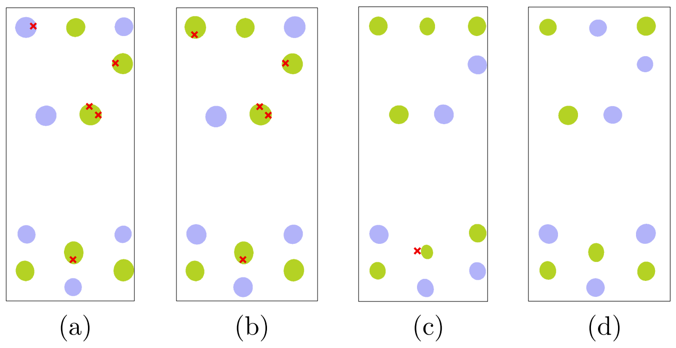


Figure 5.16: Printed image of cell `BUF_X1`; (a) [4] +[5], #EPEV=5; (b) [6] +[5], #EPEV=5; (c) [7], #EPEV=1. (d) Ours, #EPEV=0.

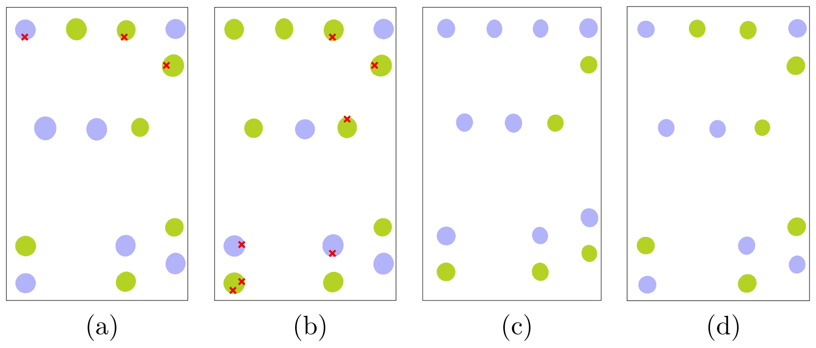


Figure 5.17: Printed image of cell `OR2_X1`; (a) [4] +[5], #EPEV=3; (b) [6] +[5], #EPEV=7; (c) [7], #EPEV=0. (d) Ours, #EPEV=0.

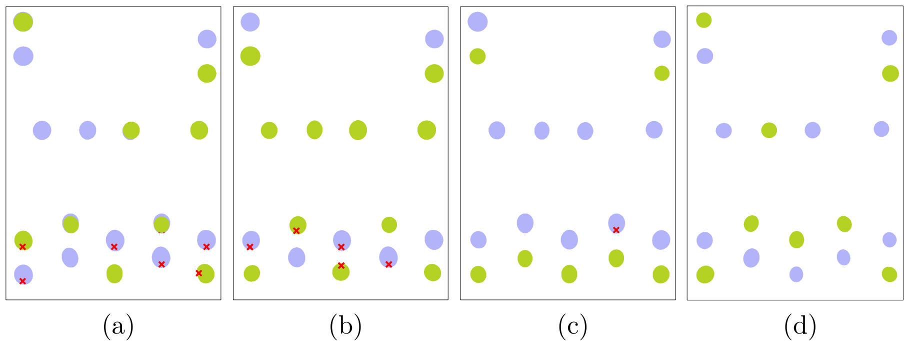


Figure 5.18: Printed image of cell NAND4_X1; (a) [4] +[5], #EPEV=6; (b) [6] +[5], #EPEV=5; (c) [7], #EPEV=1. (d) Ours, #EPEV=0.

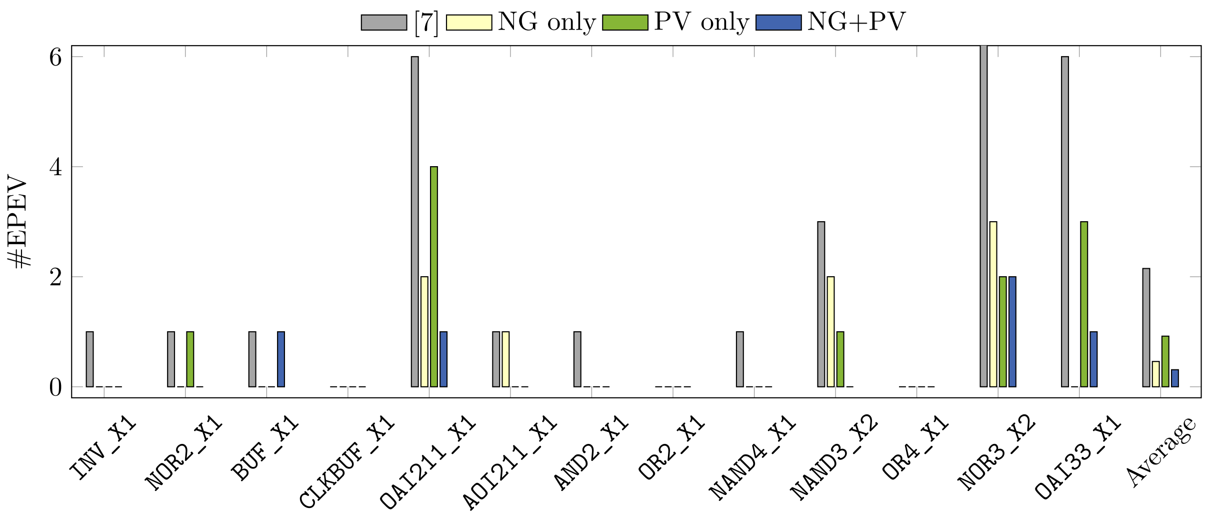


Figure 5.19: Comparison on the number of EPE violations.

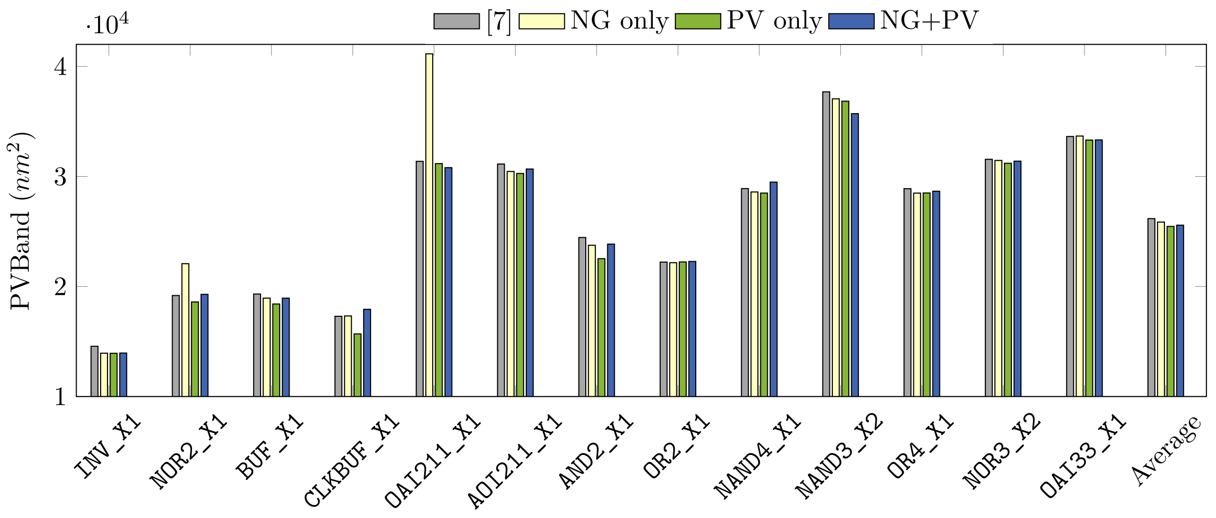


Figure 5.20: Comparison on PV Band.

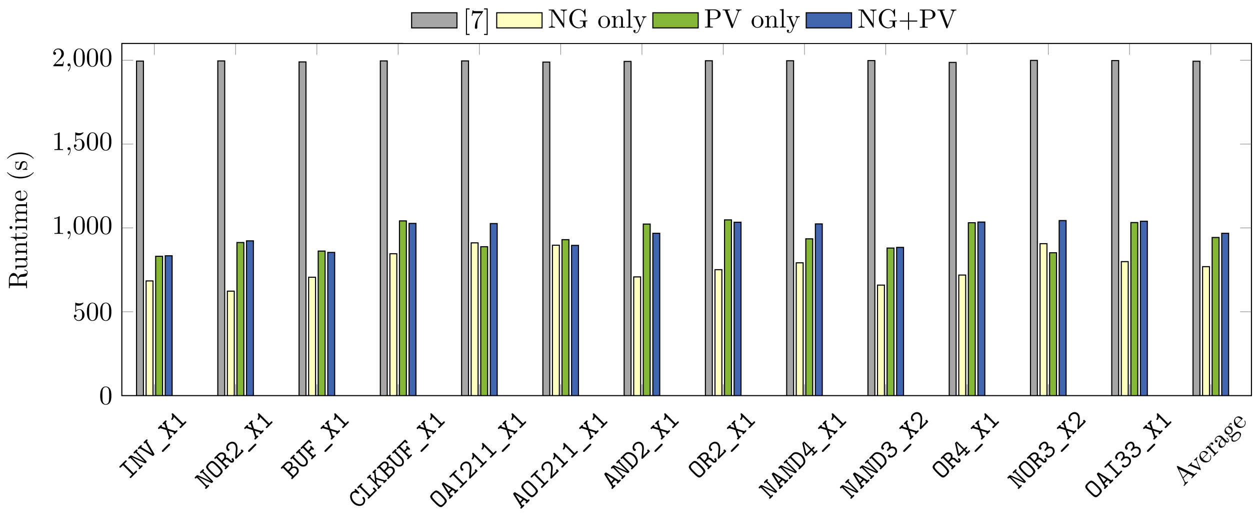


Figure 5.21: Comparison on runtime.

# Chapter 6

# Hardware-Friendly Deep Learning

Typically, the research of hardware-friendly learning can be established in different perspectives. Firstly, deep learning algorithms can be designed to be hardware-friendly in terms of storage and power consumption, which can be useful to general computing platforms. Deep neural networks are demonstrated to be over-parameterized [76], which motivates researchers to explore efficient approaches to make the deep models compact. Secondly, specific hardware can be designed for deep learning applications, as the bottleneck of processing these tasks is in the memory access. Several previous works investigated particular dataflows to maximize the data reuse, and hence the amount of DRAM accesses can be reduced significantly [142, 143, 144, 145]. In addition, a combination of hardware-software co-design is able to perform coordination between two sides [146, 147], which also draws great attention in recent years.

In this chapter, we focus on the first category and develop a set of methodologies to design efficient deep learning models for hardware-friendly learning. In Section 6.1, a unified approximation framework is developed for compressing and accelerating deep neural networks, which is a general technique and can be applied to most of commonly seen convolutional neural networks. Moreover, a pruning methodology is proposed for unsupervised partial domain adaptation in Section 6.2, which is a special scenario in deep learning applications.

## 6.1 A Unified Approximation Framework for Deep Neural Networks

Approximating the deep models involves removing the redundancy and seeking for simplified structures such that the approximated network may retain the performance on original tasks. In this work, we propose a unified approximation framework for CNNs which approximates the convolutional layers with two components, including a structured sparse component and a low-rank component. The illustration is presented in Figure 6.1. In contrast to [148], our constraints not only facilitate model compression but also favors acceleration because of the structured sparse weights [79]. We retain the accuracy of the model by approximating the nonlinear response after activation. The layer-wise network approximation problem is formulated as minimizing the reconstruction error of the response after non-linear ReLU. To overcome the resulted difficulty of non-convex optimization, we propose a convex relaxation scheme which considers the constraints for structured sparsity and low-rankness, and then solve it with an extension of alternating direction method of multipliers (ADMM) [149]. Moreover, we prove that the extended ADMM algorithm converges to the optimal solution of the relaxed problem.

The proposed method is evaluated on well-known DNN architectures, including *VGG-16* [150], *NIN* [151], *AlexNet* [152] and *GoogLeNet* [153]. For *VGG-16* with the CIFAR-10 dataset, we achieve 4.4× model compression with only 0.4% accuracy drop. Meanwhile, with the compressed model the inference is accelerated by 2.2×. For *AlexNet* with the ImageNet dataset, we achieve 4.9× model compression at the cost that the top-5 accuracy drops slightly from 81.3% to 80%. For *GoogLeNet* with the ImageNet dataset, the proposed method also brings 2.9× reduction of the model parameters without any degradation on the accuracy of inference. These experimental results reveal that the proposed approximation framework is able to remarkably compress the CNN models while keeping high accuracy.

The rest of this section is organized as follows. The problem formulation of the proposed methodology is given in Section 6.1.1. Section 6.1.2 presents a numerical optimization algorithm for solving the problem. The experimental results are reported in Section 6.2.4.

### 6.1.1 Problem Formulation

In this section, we introduce our mathematical formulation for network approximation using structured sparse and low-rank decomposition, while taking non-linearity into account. To this end, we propose to formulate the problem into a unified optimization model. In the following context, we focus on CNNs which involve a large model size.

In an FC layer of a CNN, the output feature map can be computed as

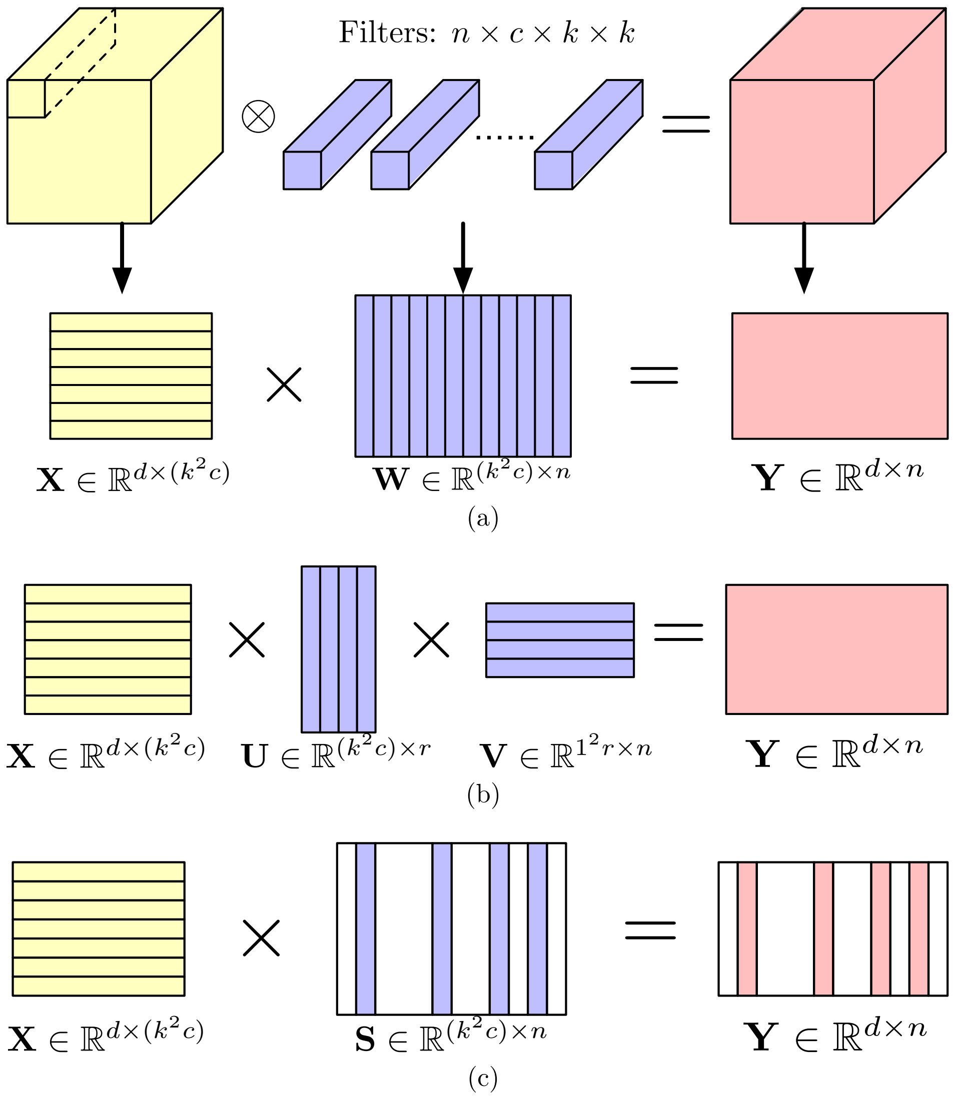
$$\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{X}, \tag{6.1}$$

where $\boldsymbol{X} \in \mathbb{R}^m$ and $\boldsymbol{Y} \in \mathbb{R}^n$ represent the input feature vector and output response, respectively. $\boldsymbol{W} \in \mathbb{R}^{n \times m}$ denotes the weight matrix. For a convolutional layer the convolution operation can also be represented as Equation (6.1). The illustration is shown in Figure 6.1a. The convolution filter is $\mathcal{W} \in \mathbb{R}^{n\times c\times k\times k}$, where $k$ is spatial size, $c$ is the number of input channels and $n$ is the number of filters. The filter can be reshaped to a matrix with size $n$-by-$k^2c$. The input $\boldsymbol{X}$ is lowered to a matrix such that each $k^2c$ volume involved in a convolution forms a column. Then the convolution operation is converted to matrix multiplication.

The information loss is inevitable when we approximate the original filters by low-rank or sparse filters, which may cause performance degradation. In order to compress the network and accelerate the computation, we perform low-rank approximation and network sparsification simultaneously. The output feature maps of a layer is generated by the sum of convolving with each filter. In order to preserve the performance, we aim at minimizing the reconstruction error of the response generated by the approximated filters in each layer after activation. An example block structure in the network is demonstrated in Figure 6.2. Then, the problem is formulated as follows:

$$\begin{aligned} &\min_{\boldsymbol{A},\boldsymbol{B}} \sum_{i=1}^{N} \left\|\boldsymbol{Y}_i - r((\boldsymbol{A}+\boldsymbol{B})\boldsymbol{X}_i)\right\|_F^2 , \\ &\text{s.t.} \quad \|\boldsymbol{A}\|_0 \le S, \quad \text{rank}(\boldsymbol{B}) \le L. \end{aligned} \tag{6.2}$$

Here $\boldsymbol{Y}_i$ and $\boldsymbol{X}_i$ represent the output feature map and the input feature map of a layer, respectively. Structured sparse component $\boldsymbol{A}$ and low-rank component $\boldsymbol{B}$ are two weight matrices we are looking for, each of which is the lowered matrices of a 4-D tensor. $N$ is the total number of samples used for approximation. $\|\cdot\|_F$ is Frobenius norm. $r(\cdot)$ is the activation function in the network, i.e., ReLU$(\cdot)$. $S$ and $L$ are user-defined target sparsity level and target rank for the filters.



Filters: $n \times c \times k \times k$
$\mathbf{X} \in \mathbb{R}^{d\times(k^2c)}$
$\mathbf{W} \in \mathbb{R}^{(k^2c)\times n}$
$\mathbf{Y} \in \mathbb{R}^{d\times n}$
(a)
$\mathbf{X} \in \mathbb{R}^{d\times(k^2c)}$
$\mathbf{U} \in \mathbb{R}^{(k^2c)\times r}$
$\mathbf{V} \in \mathbb{R}^{1^2r\times n}$
$\mathbf{Y} \in \mathbb{R}^{d\times n}$
(b)
$\mathbf{X} \in \mathbb{R}^{d\times(k^2c)}$
$\mathbf{S} \in \mathbb{R}^{(k^2c)\times n}$
$\mathbf{Y} \in \mathbb{R}^{d\times n}$
(c)

Figure 6.1: (a) Transform convolution to matrix multiplication; (b) Approximate weight matrix $\mathbf{W}$ using two matrices with lower-rank; (c) Impose structured sparsity on weight matrix $\mathbf{W}$.

### 6.1.2 Optimization Methodology

#### 6.1.2.1 Problem Relaxation

Solving Problem (6.2) directly involves both $l_0$ minimization and rank minimization, which is NP-hard. Besides, we want $\boldsymbol{A}$ to be structured sparse, which leads to extra

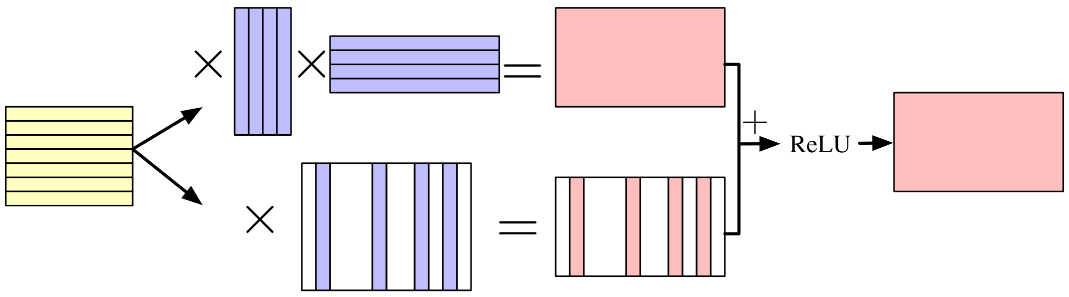


Figure 6.2: Structure combining sparse and low-rank decomposition.

difficulty. To tackle this challenge, we apply convex relaxation to the constraints. The rank constraint on $\boldsymbol{B}$ is relaxed by nuclear norm of $\boldsymbol{B}$, which is the sum of the singular values of $\boldsymbol{B}$. As for the $l_0$ norm constraints, a general way is to relax it by $l_1$ norm which is convex and has good performance in imposing sparsity. However, as we discussed above, structured sparse patterns can be more easily used for computation acceleration. Therefore, here we relax $l_0$ constraint by $l_{2,1}$ norm (the sum of the Euclidean norms of the columns) such that the zero elements in $\boldsymbol{A}$ appear column-wise. Then, the original problem is reformulated as

$$\min_{\boldsymbol{A},\boldsymbol{B}} \sum_{i=1}^{N} \|\boldsymbol{Y}_i - r((\boldsymbol{A}+\boldsymbol{B})\boldsymbol{X}_i)\|_F^2 + \lambda_1 \|\boldsymbol{A}\|_{2,1} + \lambda_2 \|\boldsymbol{B}\|_* , \tag{6.3}$$

where $\|\cdot\|_{2,1}$ is $l_{2,1}$ norm and $\|\cdot\|_*$ is nuclear norm. $\lambda_1$ and $\lambda_2$ are coefficients of the relaxed terms. The problem (6.3) now is a convex optimization problem. To solve it, we make use of the alternating direction method of multipliers (ADMM), which is widely used in large-scale problems arising in statistics [149]. Especially, the optimal solution of the sub-problems involving $l_{2,1}$-norm and nuclear norm can be obtained in closed-form as in subspace learning [154] and the singular value thresholding (SVT) operator [155], respectively.

By introducing an auxiliary variable $\boldsymbol{M}$, the Problem (6.3) can be rewritten as

$$\begin{aligned} &\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{M}} \sum_{i=1}^{N} \|\boldsymbol{Y}_i - r(\boldsymbol{M}\boldsymbol{X}_i)\|_F^2 + \lambda_1 \|\boldsymbol{A}\|_{2,1} + \lambda_2 \|\boldsymbol{B}\|_* , \\ &\text{s.t.} \;\; \boldsymbol{A} + \boldsymbol{B} = \boldsymbol{M}. \end{aligned} \tag{6.4}$$

Then the augmented Lagrangian function of Problem (6.4) is

$$\begin{aligned} L_t(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{M}, \boldsymbol{\Lambda}) &= \sum_{i=1}^{N} \|\boldsymbol{Y}_i - r(\boldsymbol{M}\boldsymbol{X}_i)\|_F^2 + \lambda_1 \|\boldsymbol{A}\|_{2,1} \\ &+ \lambda_2 \|\boldsymbol{B}\|_* + \langle \boldsymbol{\Lambda}, \boldsymbol{A} + \boldsymbol{B} - \boldsymbol{M} \rangle + \frac{t}{2} \|\boldsymbol{A} + \boldsymbol{B} - \boldsymbol{M}\|_F^2 , \end{aligned} \tag{6.5}$$

where $t > 0$ is the penalty parameter and $\boldsymbol{\Lambda}$ is Lagrange multiplier. $\langle \cdot, \cdot \rangle$ represents the inner product operator.

#### 6.1.2.2 Variables Update

ADMM solves the minimization problem of $L_t(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{M}, \boldsymbol{\Lambda})$ iteratively. The variables are alternatively updated in each iteration. To update $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{M}$ in iteration $k+1$, our algorithm takes two steps. Firstly, we consider the following three sub-problems.

$$\min_{\boldsymbol{A}} \lambda_1 \|\boldsymbol{A}\|_{2,1} + \frac{t}{2} \left\| \boldsymbol{A} + \boldsymbol{B}_k - \boldsymbol{M}_k + \frac{\boldsymbol{\Lambda}_k}{t} \right\|_F^2 , \tag{6.6}$$

$$\min_{\boldsymbol{B}} \lambda_2 \|\boldsymbol{B}\|_* + \frac{t}{2} \left\| \boldsymbol{B} + \hat{\boldsymbol{A}}_k - \boldsymbol{M}_k + \frac{\boldsymbol{\Lambda}_k}{t} \right\|_F^2 , \tag{6.7}$$

$$\begin{aligned} \min_{\boldsymbol{M}} \; &\sum_{i=1}^{N} \|\boldsymbol{Y}_i - r(\boldsymbol{M}\boldsymbol{X}_i)\|_F^2 + \langle \boldsymbol{\Lambda}_k, \hat{\boldsymbol{A}}_k + \hat{\boldsymbol{B}}_k - \boldsymbol{M} \rangle \\ &+ \frac{t}{2} \left\| \hat{\boldsymbol{A}}_k + \hat{\boldsymbol{B}}_k - \boldsymbol{M} \right\|_F^2 . \end{aligned} \tag{6.8}$$

All these three problems are proximal mapping problems. For Problem (6.6), the optimal solution is given by

$$\hat{\boldsymbol{A}}_k = \mathrm{prox}_{\frac{\lambda_1}{t}\|\cdot\|_{2,1}}(\boldsymbol{M}_k - \boldsymbol{B}_k - \frac{\boldsymbol{\Lambda}_k}{t}). \tag{6.9}$$

The explicit representation of Equation (6.9) can be derived based on [154]. Let $\boldsymbol{C} = \boldsymbol{M}_k - \boldsymbol{B}_k - \dfrac{\boldsymbol{\Lambda}_k}{t}$, then the column $i$ in $\hat{\boldsymbol{A}}_k$ is given as

$$[\hat{\boldsymbol{A}}_k]_{:,i} = \begin{cases} \dfrac{\|[\boldsymbol{C}]_{:,i}\|_2 - \frac{\lambda_1}{t}}{\|[\boldsymbol{C}]_{:,i}\|_2}[\boldsymbol{C}]_{:,i}, & \text{if } \|[\boldsymbol{C}]_{:,i}\|_2 > \dfrac{\lambda_1}{t}; \\ 0, & \text{otherwise.} \end{cases} \tag{6.10}$$

For Problem (6.7), the optimal solution is given by

$$\hat{\boldsymbol{B}}_k = \mathrm{prox}_{\frac{\lambda_2}{t}\|\cdot\|_*}(\boldsymbol{M}_k - \hat{\boldsymbol{A}}_k - \frac{\boldsymbol{\Lambda}_k}{t}). \tag{6.11}$$

The explicit representation of Equation (6.11) can be obtained based on SVT operator $\mathcal{D}_\tau$ [155]. Let $\boldsymbol{D} = \boldsymbol{M}_k - \hat{\boldsymbol{A}}_k - \dfrac{\boldsymbol{\Lambda}_k}{t}$. We perform singular value decomposition on $\boldsymbol{D}$ such that $\boldsymbol{D} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}$, where $\boldsymbol{\Sigma} = \mathrm{diag}(\{\sigma_i\}_{1\le i\le r})$ and $\sigma_i$ is the $i$-th largest singular value. Then $\hat{\boldsymbol{B}}_k$ is given by

$$\hat{\boldsymbol{B}}_k = \boldsymbol{U}\mathcal{D}_{\frac{\lambda_2}{t}}(\boldsymbol{\Sigma})\boldsymbol{V}, \tag{6.12}$$

where $\mathcal{D}_{\frac{\lambda_2}{t}}(\boldsymbol{\Sigma}) = \mathrm{diag}(\{(\sigma_i - \frac{\lambda_2}{t})_+\})$.

For Problem (6.8), it is non-trivial to derive the closed-form of the optimal solution of the sub-problem with respect to $\boldsymbol{M}$ since $r(\cdot)$ is a piecewise linear function. However, the function is continuous and convex so that we can approach the optimal solution of $\boldsymbol{M}$ iteratively by applying gradient-based method. In our implementation, we apply stochastic gradient descent (SGD) to solve it, and set learning rate as $10^{-3}$ and momentum as 0.9.

Up to now we are extending the classical ADMM to a three-block separable convex programming. This direct extension, however, is not necessarily convergent, as shown in the previous works [156, 157]. To address this issue, a simple correction step was proposed in [156], shown as follows.

$$\begin{pmatrix} \boldsymbol{B}_{k+1} \\ \boldsymbol{M}_{k+1} \\ \boldsymbol{\Lambda}_{k+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{B}_{k} \\ \boldsymbol{M}_{k} \\ \boldsymbol{\Lambda}_{k} \end{pmatrix} - \alpha \begin{pmatrix} \boldsymbol{I} & (\tau-1)\boldsymbol{I} & \boldsymbol{O} \\ \tau\boldsymbol{I} & \boldsymbol{I} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{B}_{k} - \hat{\boldsymbol{B}}_{k} \\ \boldsymbol{M}_{k} - \hat{\boldsymbol{M}}_{k} \\ \boldsymbol{\Lambda}_{k} - \hat{\boldsymbol{\Lambda}}_{k} \end{pmatrix}, \tag{6.13}$$

where $\boldsymbol{O}$ denotes zero matrix. $\tau$ is set to $\frac{1}{2}$. $\alpha$ is set to $\frac{3}{4}$. With this correction step, the extended ADMM can ensure the global convergence.

The overall optimization procedure is summarized in Algorithm 6. It starts with an initialization for all the variables and hyper-parameters (line 1). Then these variables are updated alternatively in each iteration based on the equations or SGD algorithm, as described above (line 3 – line 6). Each iteration ends up with a correction step presented as Equation (6.13) (line 7). The entire optimization procedure exits when the pre-defined condition is satisfied.

#### 6.1.2.3 Convergence Analysis

In this subsection, we prove the convergence of Algorithm 6. Let $f_1(\boldsymbol{M}) = \sum_{i=1}^{N} \|\boldsymbol{Y}_i - r(\boldsymbol{M}\boldsymbol{X}_i)\|_F^2$, $f_2(\boldsymbol{A}) = \lambda_1 \|\boldsymbol{A}\|_{2,1}$, and $f_3(\boldsymbol{B}) = \lambda_2 \|\boldsymbol{B}\|_*$. Let $\mathbf{m}$ denote the vectorization of $\boldsymbol{M}$, i.e., $\mathbf{m} = \text{vec}(\boldsymbol{M})$, and similarly, let $\mathbf{a} = \text{vec}(\boldsymbol{A})$, and $\mathbf{b} = \text{vec}(\boldsymbol{B})$.

Using these notations, the problem in Equation (6.4) takes the following generic

**Algorithm 6** ADMM for solving Problem (6.4)

```
Require: Feature maps $\boldsymbol{Y}_i, \boldsymbol{X}_i$, $i = 1 \cdots N$, given $\lambda_1$, $\lambda_2$.
Ensure: Structured sparse matrix $\boldsymbol{A}$ & low-rank matrix $\boldsymbol{B}$.
 1: Initialize $k \leftarrow 0$, $\boldsymbol{\Lambda}_0$, $\boldsymbol{A}_0$, $\boldsymbol{B}_0$, $\boldsymbol{M}_0$, error tolerance $\epsilon$, $t$;
 2: while not converged do
 3:     Calculate $\hat{\boldsymbol{A}}_k$ by Equation (6.10);
 4:     Calculate $\hat{\boldsymbol{B}}_k$ by Section 6.1.2.2;
 5:     Calculate $\hat{\boldsymbol{M}}_k$ by solving Problem (6.8) with SGD method;
 6:     $\hat{\boldsymbol{\Lambda}}_k \leftarrow \boldsymbol{\Lambda}_k + t(\hat{\boldsymbol{A}}_k + \hat{\boldsymbol{B}}_k - \hat{\boldsymbol{M}}_k)$;
 7:     Perform correction step by Equation (6.13);
 8:     $k \leftarrow k + 1$;
 9: end while
10: return $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$;
```

form

$$\begin{aligned} &\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{M}} f_1(\boldsymbol{M}) + f_2(\boldsymbol{A}) + f_3(\boldsymbol{B}), \\ &\text{s.t.} \quad \boldsymbol{C}_1\mathbf{a} + \boldsymbol{C}_2\mathbf{b} - \boldsymbol{C}_3\mathbf{m} = \mathbf{c}, \end{aligned} \tag{6.14}$$

where $\boldsymbol{C}_1$, $\boldsymbol{C}_2$, and $\boldsymbol{C}_3$ are the identity matrices, and $\boldsymbol{c} = \mathbf{0}$. The convergence of ADMM for solving the standard form (6.14) was studied in [156, 157]. We establish the convergence of our algorithm by transforming the problem in Equation (6.4) into a standard form (6.14). Note that our algorithm alternates between three blocks of variables, $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{M}$. According to the definitions of $f_1(\boldsymbol{M})$, $f_2(\boldsymbol{A})$, and $f_3(\boldsymbol{B})$, it is easy to verify the problem in Equation (6.4) and our algorithm satisfy the convergence conditions of the problem in Equation (6.14), as stated in [156]. Thus, we have the following theorem.

**Theorem 1** *Consider the problem in Equation* (6.4)*, where $f_1(\boldsymbol{M})$, $f_2(\boldsymbol{A})$, and $f_3(\boldsymbol{B})$, are convex functions, and $\boldsymbol{C}_1$, $\boldsymbol{C}_2$, and $\boldsymbol{C}_3$ are the identity matrices, and have full column* *rank. The sequence* $\{\boldsymbol{A}_k, \boldsymbol{B}_k, \boldsymbol{M}_k\}$ *generated by Algorithm 6 converges to the optimal solution* $\{\boldsymbol{A}^*, \boldsymbol{B}^*, \boldsymbol{M}^*\}$ *of the problem in Equation* (6.4).

### 6.1.3 Experimental Results

#### 6.1.3.1 Experimental Setup

Algorithm 6 takes in the input and output feature maps generated from the inference on some sample data, and outputs the approximated network layers. The tested CNNs include *VGG-16* [150], *NIN* [151], *AlexNet* [152] and *GoogLeNet* [153]. For each network, we first obtain its approximation, and then fine-tune the network based on the obtained structures to restore the accuracy. During the approximation, different layers use different weight coefficients $\lambda_1$ and $\lambda_2$. In our experiments, we find out setting $\lambda_2$ to be $2.5 \sim 3$ times larger than $\lambda_1$ gives good trade-off between accuracy and model compression rate. And we let $\lambda_1$ ranges from $0.08 \sim 0.3$. The penalty parameter $t$ in (5) is set to $10^{-3}$. The runtime of Algorithm 6 varies from layer to layer, ranging from 10 minutes to half an hour.

The inference is conducted on Caffe [158] using CIFAR-10 and ILSVRC-2012, i.e., ImageNet [159]. After the network approximation, a small initial learning rate of $10^{-5}$ is used in the fine-tuning step. We use three metrics for evaluation, including accuracy loss, compression rate (CR) and speedup ratio. The CR is calculated as

$$\mathrm{CR} = \frac{\text{Approximated layer size}}{\text{Original layer size}} \times 100\%. \tag{6.15}$$

The accuracy loss is the degradation on accuracy after approximation, denoted by "*accu.* $\downarrow$" in the table. The "speed-up" ratio indicates the acceleration for inference.

Table 6.1: Results on *VGG-16* with CIFAR-10

| Layer | $CR(\boldsymbol{A})(\%)$ | $CR(\boldsymbol{B})(\%)$ | $CR(\boldsymbol{A}+\boldsymbol{B})(\%)$ |
|---|---|---|---|
| `conv1-1` | 0.0 | 100.0 | 100 |
| `conv1-2` | 5.4 | 52.1 | 57.5 |
| `conv2-1` | 5.4 | 38.2 | 43.6 |
| `conv2-2` | 2.2 | 28.6 | 30.8 |
| `conv3-1` | 2.8 | 47.7 | 50.5 |
| `conv3-2` | 4.0 | 54.3 | 58.3 |
| `conv3-3` | 10.0 | 59.0 | 69.0 |
| `conv4-1` | 2.0 | 16.0 | 18.0 |
| `conv4-2` | 2.0 | 22.4 | 24.4 |
| `conv4-3` | 4.0 | 16.3 | 24.3 |
| `conv5-1` | 2.0 | 9.8 | 11.8 |
| `conv5-2` | 2.0 | 8.7 | 10.7 |
| `conv5-3` | 2.0 | 6.7 | 8.7 |
| `fc1` | 44.2 | 0.0 | 44.2 |
| `fc2` | 36.2 | 0.0 | 36.2 |
| `fc3` | 24.0 | 0.0 | 24.0 |
| CR | 22.5% | (4.44× reduction of model size) | |
| Speed-up | | 2.2× | |
| Accu. ↓ | | 0.40% | |

#### 6.1.3.2 Experiments on *CIFAR-10*

**VGG-16** *VGG-16* [150] network is a convolutional neural network consisting of 13 convolution layers and 3 FC layers. All the convolutional filters have the same spatial size of $3 \times 3$. We test the proposed method with experiments on the CIFAR-10 dataset which consists of 50K training images and 10K test images. We first train a *VGG-16* network from scratch to obtain the baseline, which has an accuracy of 92.05%. To make the approximation, 1000 images are selected from training set for inference and the input and output feature maps are collected for Algorithm 6.

The approximation is performed on each layer sequentially. The layer-wise approx-

Table 6.2: Results on *NIN* with CIFAR-10

| Layer | $CR(\boldsymbol{A})(\%)$ | $CR(\boldsymbol{B})(\%)$ | $CR(\boldsymbol{A}+\boldsymbol{B})(\%)$ |
|---|---|---|---|
| `conv1` | 0.0 | 18.4 | 18.4 |
| `cccp1` | 0.0 | 100.0 | 100 |
| `cccp2` | 0.0 | 100.0 | 100 |
| `conv2` | 0.0 | 16.9 | 16.9 |
| `cccp3` | 0.0 | 100.0 | 100 |
| `cccp4` | 0.0 | 100.0 | 100 |
| `conv3` | 0.0 | 38.2 | 38.2 |
| `cccp5` | 0.0 | 100.0 | 100 |
| `cccp6` | 0.0 | 100.0 | 100 |
| CR | 36.0% | (2.77× reduction of model size) | |
| Speed-up | | 2.2× | |
| Accu. $\downarrow$ | | 0.41% | |

imation results are shown in Table 6.1. In our experiment, we find out approximating the first convolutional layer may lead to significant accuracy drop. Therefore, the first layer is not approximated. The sparse component $\boldsymbol{A}$ is stored in CSR format. Moreover, we constrain the sparse component $\boldsymbol{A}$ to be structured sparse to accelerate the computation as in [79]. The low-rank component is represented by the product of two smaller matrices. For FC layers, we only use the sparse component for approximation to reduce accuracy drop.

The performance comparison with other previous work [78] is presented in Table 6.3. With the approximation, the model size is reduced by 4.44×, which corresponds to 2.2× speedup on inference. Both compression rate and speedup ratio outperform [78]. Without fine-tuning, there is some classification accuracy drop. In order to restore the accuracy of the compressed model, we retrain the compressed network with the training set for 5 epochs. With this fine-tuning step the accuracy loss reduces from 1.8% to only

Table 6.3: Comparison on CIFAR-10

| Model | Method | Accu. ↓ | CR | Speed-up |
|---|---|---|---|---|
| VGG-16 | Original | 0.00% | 1.00 | 1.00 |
| | ICLR'17 [78] | **0.06%** | 2.70 | 1.80 |
| | Ours | 0.40% | **4.44** | **2.20** |
| NIN | Original | 0.00% | 1.00 | 1.00 |
| | ICLR'16 [86] | 1.43% | 1.54 | 1.50 |
| | IJCAI'18 [85] | 1.43% | 1.45 | - |
| | Ours | **0.41%** | **2.77** | **1.70** |

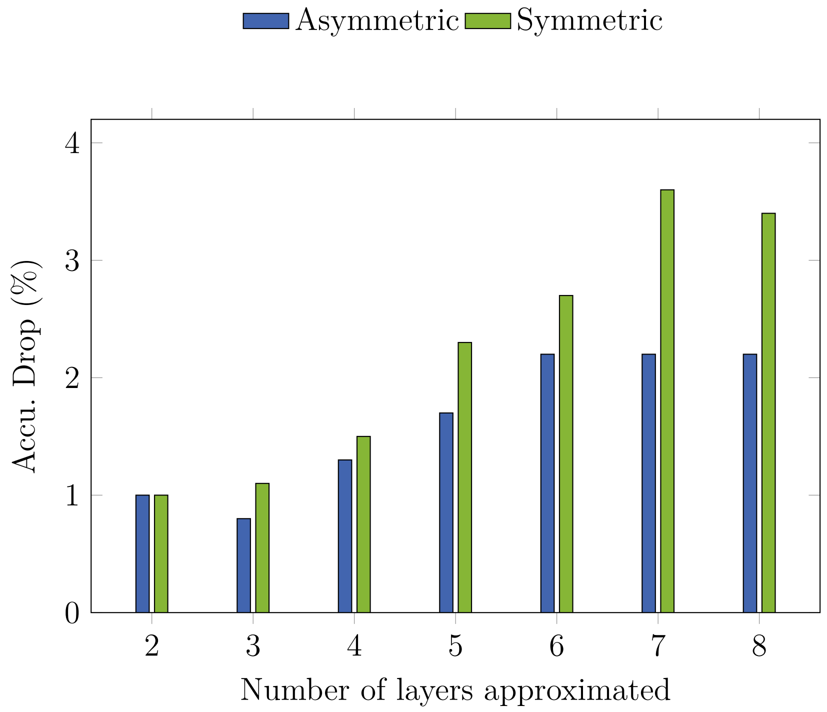
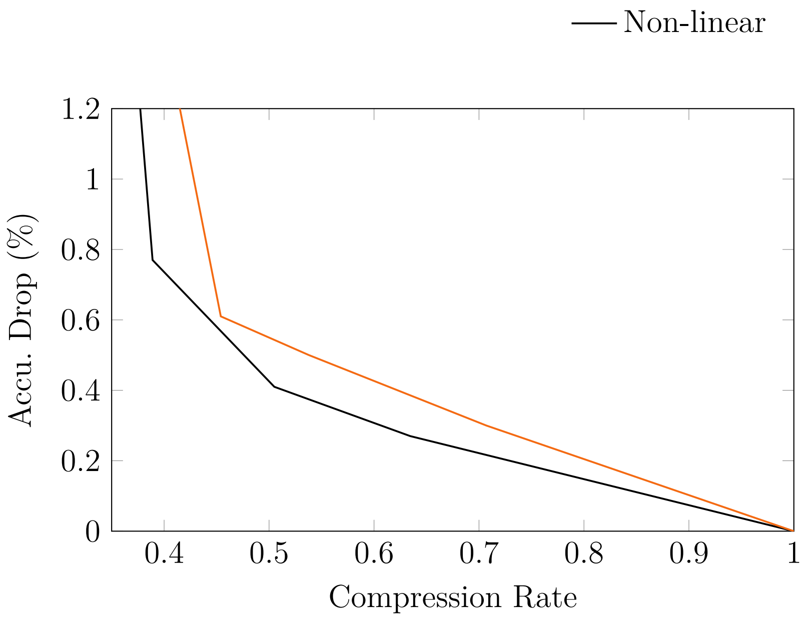
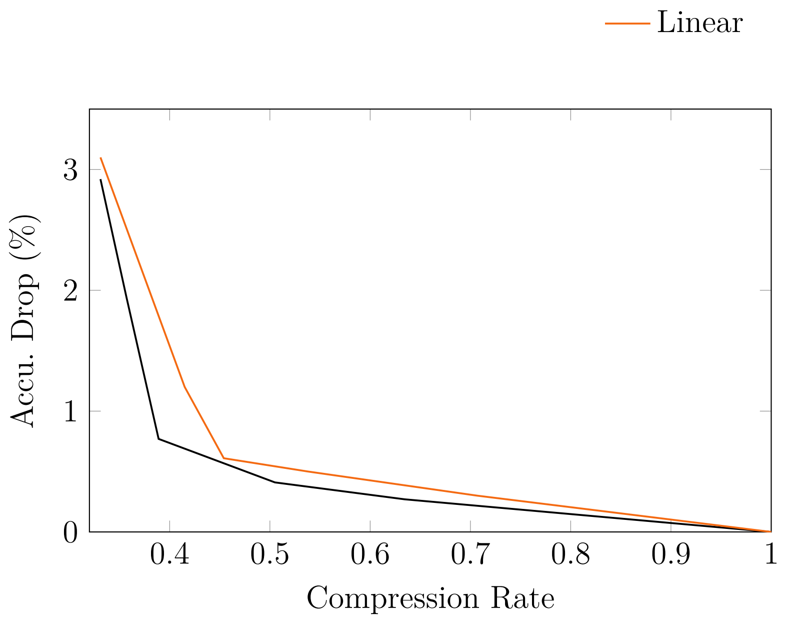


Figure 6.3: Accuracy on CIFAR-10 using symmetric reconstruction and asymmetric reconstruction.

0.40%, which becomes very close to the accuracy of the original VGG-16.



(a)



(b)

Figure 6.4: Comparison of reconstructing linear response and non-linear response: (a) layer `conv2-1`; (b) layer `conv3-1`.

If a shallow layer is approximated, the approximation error may be accumulated when deeper layers are approximated. In order to handle this issue, we take the 'asymmetric' strategy used in [160]. We approximate the layers from shallow to deep. When approximating a deep layer, use the response produced by all previous layers instead of the non-approximate response as the input feature map $\boldsymbol{X}_i$. Figure 6.3 shows the comparison of classification error increase. We can observe that with more layers being approximated, the performance becomes worse for both strategies. However, the asymmetric version loses less accuracy.

We further compare the performance between reconstructing non-linear response and reconstructing linear response. We perform the comparison on a single layer each time, while the remaining layers are kept unchanged. In Figure 6.4, we plot the relation between the CR and the accuracy degradation of two approaches of different layers. The performance is evaluated by the accuracy drop compared with original model. We take two convolutional layers in two different stages of the *VGG-16*, including `conv2-1` and `conv3-1`. Figure 6.4 shows that under the same CR, reconstructing non-linear response achieves lower accuracy drop than reconstructing linear response, which verifies the advantage of reconstructing the non-linear response. In Figure 6.5, we visualize the sparse filter and low-rank filter after the approximation of layer `conv3-1`. $\boldsymbol{B}$ has rank 136 and it can be further decomposed by $\boldsymbol{B} = \boldsymbol{U}\boldsymbol{V}$, where both $\boldsymbol{U}$ and $\boldsymbol{V}$ have rank 136.

**NIN** Network-in-network (*NIN*) [151] has 9 convolutional layers among which 6 layers have a spatial size of $1 \times 1$. Considering that these $1 \times 1$ convolutional layers have less contribution to the overall model size and computation, we focus on remaining three layers which have spatial size of $3 \times 3$ or $5 \times 5$. We present the layer-wise approximation

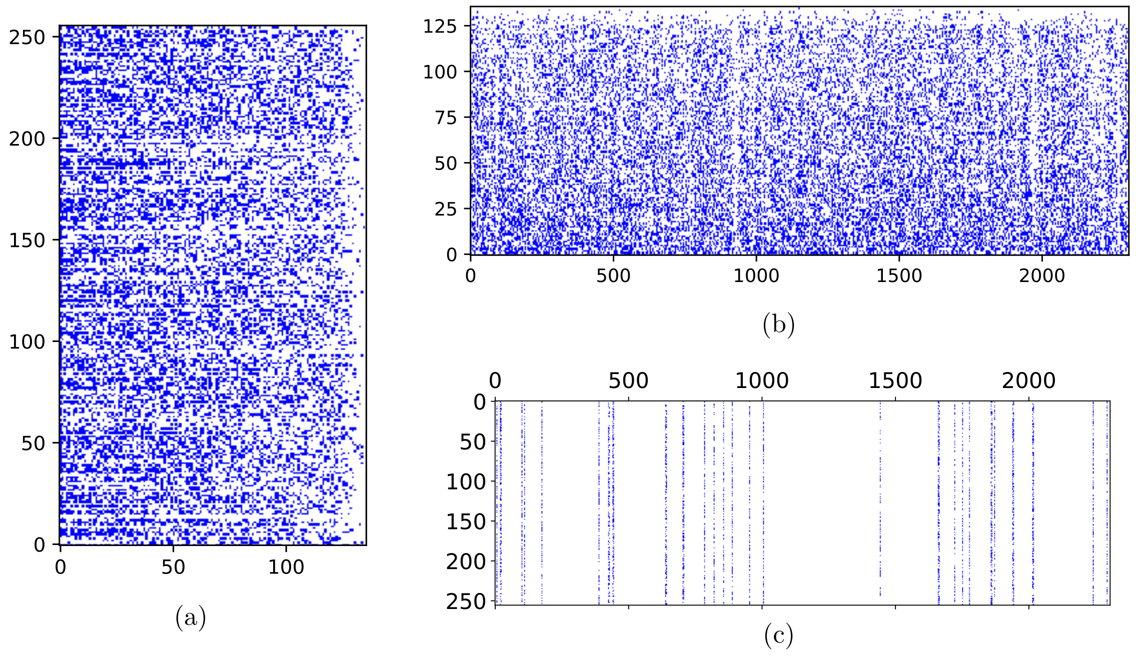


Figure 6.5: Approximated filters of `conv3-1`. Blue dots have non-zero values. Low-rank filter $\boldsymbol{B}$ with rank 136 is decomposed into $\boldsymbol{UV}$, both of which have rank 136. (a) Matrix $\boldsymbol{U}$; (b) Matrix $\boldsymbol{V}$. (c) Column-wise sparse filter $\boldsymbol{A}$.

results in Table 6.2. It can be observed that for all the approximated convolutional layers, only low-rank component is used and structured sparse didn't show up, which means approximating *NIN* using CIFAR10 dataset reduces to low-rank approximation and sparse components are not beneficial to the objectives. It indicates that the proposed unified framework is flexible to find good solutions and does not rely on prior assumptions to achieve good results.

Experimental results using the same network (i.e., *NIN*) and CIFAR10 are reported in previous work [86, 85]. The comparison of accuracy loss, compression rates and the accuracy is shown in Table 6.3. We can see that the number of parameters is reduced by 2.77× and the inference time is accelerated by 1.70×, with only 0.41% accuracy loss compared with original model. All these three metrics are significantly better than previous work [86, 85].

#### 6.1.3.3 Experiments on *ImageNet*

**AlexNet** *AlexNet* [152] has 5 convolutional layers and 3 FC layers. It is tested for the ImageNet classification task. We evaluate the top-5 accuracy with single-view. The ILSVRC-2012 dataset consists of 1.2 million training images and 50 thousand test images. Images are resized with 256 pixels on the shorter side. The testing image is on the center crop of $224 \times 224$ pixels. We use the pre-trained model provided by Caffe Model Zoo as the baseline. In our experiment, we first select 1500 images from the training set and collect their responses for building the approximate network. The layer-wise approximation results are demonstrated in Table 6.4. The first two convolutional layers of *AlexNet* are not approximated, in order to preserve good accuracy. For FC layers, again we only use the structured sparse component for approximation.

The compression rates and the accuracy comparison are shown in Table 6.6. From the table we see that the network is compressed by more than $5\times$, which outperforms [86], [161], and [81], while the top-5 accuracy drop is only 1.3%. This reveals that the proposed approximation framework can remarkably compress *AlexNet* while keeping good accuracy.

**GoogLeNet** *GoogLeNet* [153] is another widely used network in image recognition and classification. Different from *AlexNet*, *GoogLeNet* combines two spatial sizes of convolutional filters, $3 \times 3$ and $5 \times 5$, in each inception block. In order to collect the input samples for optimization, we use a pre-trained model provided by Caffe Model Zoo to perform inference and dump the input and output feature maps of each convolutional layer. After performing approximation on *GoogLeNet*, both model size and inference

Table 6.4: Results on *AlexNet* with ILSVRC-2012

| Layer | $CR(\boldsymbol{A})(\%)$ | $CR(\boldsymbol{B})(\%)$ | $CR(\boldsymbol{A}+\boldsymbol{B})(\%)$ |
|---|---|---|---|
| `conv1` | 0.0 | 100.0 | 100.0 |
| `conv2` | 21.4 | 23.6 | 45.0 |
| `conv3` | 30.0 | 22.9 | 52.9 |
| `conv4` | 0.0 | 32.6 | 32.6 |
| `conv5` | 0.0 | 26.0 | 26.0 |
| `fc1` | 12.8 | 0.0 | 12.8 |
| `fc2` | 26.2 | 0.0 | 26.2 |
| `fc3` | 18.8 | 0.0 | 18.8 |
| CR | 18.0% | (5.56× reduction of model size) | |
| Speed-up | | 1.1× | |
| Top-5 accu. ↓ | | 1.27% | |

time are reduced. The layer-wise approximation results are shown in Table 6.5. The comparison of accuracy loss, compression rates and accuracy are shown in Table 6.6. We can see that the model size is reduced by 2.87× and the inference time is accelerated by 1.35×, without loss on accuracy. All these three metrics are significantly better than previous works using the same network model and dataset [86, 161, 81].

### 6.1.4 Summary

In this work, we have proposed a unified approximation model for DNNs with simultaneous low-rank approximation and structured sparsification. It also considers the non-linear activation to retain the accuracy. To obtain this model, a layer-wise optimization problem is presented, relaxed, and solved with an extended ADMM algorithm whose convergence is provably guaranteed. The effectiveness of the proposed approximation framework is verified on *VGG-16*, *NIN*, *GoogLeNet* and *AlexNet*. By sacrificing little accuracy, *VGG-16* and *AlexNet* are compressed by up to 5.56×. *GoogLeNet* is

Table 6.5: Results on *GoogLeNet* with ILSVRC-2012

| Layer | $CR(\boldsymbol{A})(\%)$ | $CR(\boldsymbol{B})(\%)$ | $CR(\boldsymbol{A}+\boldsymbol{B})(\%)$ |
|---|---|---|---|
| `conv1` | 0.0 | 100.0 | 100.0 |
| `conv2` | 0.0 | 100.0 | 100.0 |
| `inception-3a` | 2.1 | 31.1 | 33.2 |
| `inception-3b` | 5.4 | 39.8 | 45.3 |
| `inception-4a` | 3.8 | 28.6 | 32.4 |
| `inception-4b` | 2.3 | 23.7 | 26.1 |
| `inception-4c` | 8.9 | 29.9 | 38.9 |
| `inception-4e` | 2.7 | 23.7 | 26.5 |
| `inception-5a` | 2.4 | 28.8 | 31.2 |
| `inception-5b` | 1.6 | 31.6 | 33.3 |
| `fc` | 35.0 | 0.0 | 35.0 |
| CR | 34.8% | (2.87× reduction of model size) | |
| Speed-up | | 1.35× | |
| Top-5 accu. ↓ | | 0.00% | |

compressed by nearly 3× without loss of accuracy. What's more, since structured sparse filters and low-rank filters are independent to each other, more inference speedup may be expected if taking actual architecture and parallel computing into account.

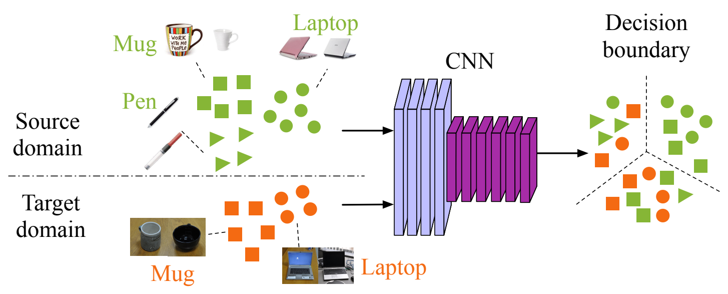
Table 6.6: Comparison on ILSVRC-2012

| Model | Method | Top-5 Accu.↓ | CR | Speed-up |
|---|---|---|---|---|
| AlexNet | Original | 0.00% | 1.00 | 1.00 |
| | ICLR'16 [86] | **0.37%** | 5.00 | **1.82** |
| | ICLR'16 [161] | 1.70% | 5.46 | 1.81 |
| | CVPR'18 [81] | 1.43% | 1.50 | - |
| | Ours | 1.27% | **5.56** | 1.10 |
| GoogleNet | Original | 0.00% | 1.00 | 1.00 |
| | ICLR'16 [86] | 0.42% | 2.84 | 1.20 |
| | ICLR'16 [161] | 0.24% | 1.28 | 1.23 |
| | CVPR'18 [81] | 0.21% | 1.50 | - |
| | Ours | **0.00%** | **2.87** | **1.35** |

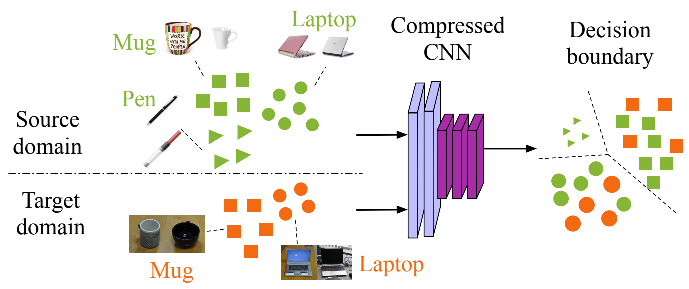
## 6.2 Compressing Deep Neural Networks for Partial Domain Adaptation

A common restriction of conventional DL algorithms is the great demand of labeled data which is costly and hence hinders the deployment of DL. Domain adaptation is a promising solution to tackle this problem, which leverages rich labeled data in the source domain to build a model for the target domain where very limited or even no labeled data is available. The core idea is to learn domain invariant representations such that the cross-domain distribution inconsistency can be resolved. Thanks to the extraordinary capability of feature extraction of deep models, recent studies show that deep learning models can achieve compelling performance in domain adaptation [98, 93, 94, 162, 97].

Despite the appealing performance of deep learning models on partial domain adaptation, the execution overhead remains a critical issue for modern deep neural networks in terms of power consumption and storage occupation. Intuitively, the number of parameters in a neural network suggests its representation capability. Therefore, it is worth exploring model compression for partial domain adaptation since redundancy is more likely to exist in this situation. For example, a large CNN is designed and trained on a large/difficult labeled dataset (e.g., ImageNet-1000), and it needs to be transferred to a small/easy dataset (e.g., Office-31). There is a high chance that the original model is over parameterized for the target task, which motivates us to compress the model. Although there is rich literature studying model compression, most of them are developed for supervised learning tasks, in which labeled data is needed to guide the pruning process or retrain a pruned model to retain performance. Unfortunately, these methods



(a)



(b)

Figure 6.6: (a) Conventional MMD-based approach applied to partial domain adaptation. Outlier class (represented by triangle) leads to poor performance; (b) Proposed approach. Increase the importance of shared classes and decrease the importance of outlier classes, with the model size reduced at the same time.

cannot be directly applied to our domain adaptation scenario due to the unavailability of labeled data in the target domain.

In this work, we investigate a new perspective for the PDA problem which is combined with model compression. The model compression and PDA are seamlessly integrated into a unified training process by iteratively pruning and training a base network. As a result, a slimmed model is obtained and negative transfer can be circumvented. The resulting model also achieves superior performance than general maximum mean discrepancy (MMD) approaches, as shown in Figure 6.6. Specifically, we design two collaborative schemes for network training and pruning, respectively. On one hand, the distribution discrepancy is bridged by minimizing a soft-weighted MMD to learn the domain-invariant features and promote the knowledge transfer, which is more effective for the PDA problem than the hard-weighted scheme. The class weights can be directly computed for the source domain because the labels are available. By assigning a soft pseudo label to each sample in the target domain, the class weights are estimated for the target domain. Based on the class weights in both domains, the shared classes and outlier classes can be distinguished and a soft-weighted MMD is calculated and adopted in training. On the other hand, a channel pruning scheme is designed on top of the network training. The importance of each channel is evaluated and those less important channels are identified and pruned. In contrast to other methods that directly evaluate on channels, we leverage the scaling factors in batch normalization (BN) layers for channel pruning based on Taylor expansion, which makes use of the gradient statistics during backward computation, thus pruning can be naturally integrated with model training. Reducing the model size also reduces the chance of over-fitting, hence our proposed model can achieve appealing performance on the target task and is energy efficient for deployment. In summary, the main contributions of this work are as follows.

- The partial domain adaptation problem is investigated from the model compression perspective using a unified training and pruning process;
- Domain discrepancy issue in the PDA problem is addressed by a soft-weighted MMD. It can omit outlier samples in contrast to conventional MMD, and is beneficial to training convergence compared to hard-weighted MMD;
- Model pruning is performed with BN scaling factors based on Taylor expansion, which can be naturally integrated into model training;

- Experimental results demonstrate that the proposed method can reduce both computation and model size by more than 70% with little performance degradation compared to state-of-the-art PDA methods.

### 6.2.1 Unsupervised Partial Domain Adaptation with Soft-weighted MMD

Unsupervised domain adaptation is a challenging task because the labels in the target domain are not available. Firstly, we briefly introduce a conventional MMD metric, which is used to represent the distance between distributions and is widely used in previous works for unsupervised domain adaptation [93, 162, 104]. Given the samples from source domain $\mathcal{D}_s$ and target domain $\mathcal{D}_t$, MMD can be empirically estimated as follows [163]:

$$\mathrm{MMD}^2(\mathcal{D}_s, \mathcal{D}_t) = \left\| \frac{1}{n_s} \sum_{\mathbf{x}_i \in \mathcal{D}_s} \phi(\mathbf{x}_i) - \frac{1}{n_t} \sum_{\mathbf{x}_j \in \mathcal{D}_t} \phi(\mathbf{x}_j) \right\|_{\mathcal{H}}^2, \tag{6.16}$$

where $\mathcal{H}$ denotes Reproducing Kernel Hilbert Space (RKHS), and $\phi(\cdot)$ denotes the feature mapping from samples to RKHS and is associated with Gaussian kernel. $n_s$ and $n_t$ represent the number of samples in source domain and target domain, respectively.

Previous works apply MMD based on an assumption that the label space is fully shared between source and target domains. However, in the case of PDA where the assumption does not hold, the MMD-based methods cannot be directly applied. More specifically, the class prior distributions are significantly different between source and target domains since a certain number of classes do not even exist in the target domain. To make MMD effective, we need to distinguish if a sample $\mathbf{x}_i \in \mathcal{D}_s$ belongs to shared classes or outlier classes, and rely on the samples in shared classes for knowledge transfer. However, it is not easy since it is unknown to us which categories are shared. To handle this, we design a weighting mechanism on class-level to identify the shared classes and the outlier classes, which can be converted to instance-level and leads to a weighted MMD criterion to tackle the PDA problem. A related approach is studied to address the class bias [162] by re-weighting classes in label space. PDA can be seen as an extreme case of class bias.

Let $\mathcal{Y}_s$ and $\mathcal{Y}_t$ denote the label space of the source domain and the target domain, respectively. Then $\mathcal{Y}_t \subset \mathcal{Y}_s$ is the condition in partial domain adaptation. Denote the weights of classes as a vector $\mathbf{w} \in \mathbb{R}^{|C_s|}$. Let $w_c^{(s)}$ and $w_c^{(t)}$ denote the weight of class $c \in \mathcal{Y}_s$ in the source domain and the target domain, respectively. Since the labels of the source domain are available, we can obtain the number of samples for each class $c$ in source domain, denoted by $n_c^{(s)}$, then the weight of class $c$ in source domain is calculated as $w_c^{(s)} = \frac{n_c^{(s)}}{n^{(s)}}$, where $n^{(s)}$ is the total number of samples in source domain. Note $w_c^{(t)} = 0$ for $c \in \mathcal{Y}_s \setminus \mathcal{Y}_t$. Let $r_c = \frac{w_c^{(t)}}{w_c^{(s)}}$. Given a set of samples $\{(\mathbf{x}_i^{(s)}, y_i^{(s)})\}$ drawn from source domain and $\{\mathbf{x}_j^{(t)}\}$ drawn from target domain, the weighted MMD is empirically estimated as:

$$\mathrm{WMMD}^2(\mathcal{D}_s, \mathcal{D}_t) = \left\| \frac{1}{\sum\limits_{\mathbf{x}_i \in \mathcal{D}_s} r_{y_i^{(s)}}} \sum_{\mathbf{x}_i \in \mathcal{D}_s} r_{y_i^{(s)}} \phi(\mathbf{x}_i) - \frac{1}{n_t} \sum_{\mathbf{x}_j \in \mathcal{D}_t} \phi(\mathbf{x}_j) \right\|_{\mathcal{H}}^2 . \qquad (6.17)$$

Typically, MMD is implemented as a loss layer in the network and integrated with the training. The CEM framework [164] is applied to network training in [162]. In each training iteration, a hard pseudo label $y_j'$ is assigned to each sample $\mathbf{x}_j \in \mathcal{D}_t$. An estimation of the class prior in the target domain is obtained based on the percentage of each class. However, it did not work well in PDA. The reasons are two-fold. (1) Assigning hard pseudo labels to the samples $\mathbf{x}_j \in \mathcal{D}_t$ makes the process easily get stuck at the local optima if the hard labels are wrong from the very beginning, and it is very likely to happen since the network is not well-trained and hence not discriminative enough; (2) Due to the missing categories in $\mathcal{Y}_t$, several elements in $\mathbf{r}$ could go to 0, which is equivalent to omitting part of the samples $\mathbf{x}_i \in \mathcal{D}_s$ when calculating the weighted MMD based on Equation (6.17). In this case, the number of effective samples to calculate MMD in each iteration may be very small, which makes MMD not an effective estimation for distribution discrepancy if combined with the reason (1), thus leading to inferior results. Detailed statistics and results will be presented in Section 6.2.4.

To address this issue, we assign soft pseudo labels instead of hard labels to the target samples, based on which a soft-weighted MMD (SWMMD) is calculated as a regularization for network training. In contrast to assigning a specific class $y'_j \in \mathcal{Y}_s$ to a target sample $\mathbf{x}_j \in \mathcal{D}_t$, the soft pseudo label is obtained by using the softmax function to compute the posterior distribution based on the output of the last fully connected layer in the network. Then the class weights of target domain $\mathbf{w}^{(t)} \in \mathbb{R}^{|\mathcal{Y}_s|}$ is estimated by averaging the soft pseudo labels over all the samples $\mathbf{x}_j \in \mathcal{D}_t$. Denote the parameters in the network as $\mathbf{W}$ and the entire forward computation as $f(\mathbf{W}, \cdot)$, $\mathbf{w}^{(t)}$ can be calculated as:

$$\mathbf{w}^{(t)} = \frac{1}{n_t}\sum_{j=1}^{n_t} \tilde{\mathbf{y}}_j = \frac{1}{n_t}\sum_{j=1}^{n_t} \text{Softmax}(f(\mathbf{W}, \mathbf{x}_j)), \tag{6.18}$$

where $\tilde{\mathbf{y}} \in \mathbb{R}^{|\mathcal{Y}_s|}$ is the posterior predictive distribution, i.e., soft pseudo label, of a sample $\mathbf{x}_j \in \mathcal{D}_t$. Then the soft label-based class weights $\mathbf{r} \in \mathbb{R}^{|\mathcal{Y}_s|}$ is calculated as mentioned before. Recall that the weighted MMD can be calculated as Equation (6.17) as long as the class weights are provided. Therefore, SWMMD can be similarly calculated based on Equation (6.17) using the soft label-based class weights $\mathbf{r}$.

The loss function of the network contains a supervised classification loss $\mathcal{L}_{cl}$ on the source domain data $\{(\mathbf{x}_i^{(s)}, y_i^{(s)})\}$ and SWMMD (Equation (6.17)) as regularization. The classification loss is formulated as:

$$\mathcal{L}_{cl} = -\frac{1}{n_s} \sum_{\mathbf{x}_i \in \mathcal{D}_s} \sum_{c=1}^{|\mathcal{Y}_s|} y_{i,c}^{(s)} \log(\tilde{y}_{i,c}), \tag{6.19}$$

where $\tilde{\mathbf{y}}_i$ and $y_i^{(s)}$ is the prediction posterior distribution and the one-hot encoded label of data $\mathbf{x}_i$, and $\tilde{y}_{i,c}$ is the predicted probability of being class $c$.

In addition, an entropy minimization principle [165] is included, which is to minimize the entropy over the posterior predictive probability of the samples on target domain and is formulated as

$$\mathcal{L}_{en} = \frac{1}{n_t} \sum_{\mathbf{x}_j \in \mathcal{D}_t} (-\sum_{c=1}^{|\mathcal{Y}_s|} \tilde{y}_{j,c} \log \tilde{y}_{j,c}). \tag{6.20}$$

Therefore, the training loss is written as

$$\mathcal{L} = \mathcal{L}_{cl} + \beta \cdot \mathrm{SWMMD}^2(\mathcal{D}_s, \mathcal{D}_t) + \gamma \cdot \mathcal{L}_{en}. \tag{6.21}$$

During the training, the class weights $\mathbf{r}$ and model parameters $\mathbf{W}$ are alternatively updated. $\mathbf{r}$ is updated at the start of each iteration using the up-to-date parameters $\mathbf{W}$ in the network. Then the loss is calculated (Equation (6.21)) and back-propagated to update model parameters $\mathbf{W}$.

### 6.2.2 Model Pruning for Unsupervised Partial Domain Adaptation

In addition to achieving high performance on the target tasks, the execution overhead should also be taken into consideration since redundancy is highly speculated to exist in base networks of the PDA model. To ease the overhead and remove redundancy in the network for PDA, a channel pruning method is introduced for a complement. Network pruning is to remove the parameters in the network which are useless or even harmful to the target task, and hence lead to a sparsified neural network. Several previous works utilize a mask to the network and each value in the mask serves as a scaling factor of a feature map. Then sparsifying the network is equivalent to sparsifying the mask. The mask can be trained along with the weights in the network with certain sparsity driven regularizations [166, 167, 168, 169]. As a result, extra parameters are introduced as masks to the training stage.

Batch normalization (BN) [170] is already widely applied in conventional DNNs to facilitate the training process. In this work, we adopt a pruning scheme by exploiting the statistics in BN layers to directly prune redundant channels without introducing extra training weights. Regarding the PDA task, it is revealed in AdaBN [171] that BN layers contain the traits of the data domain, which suggests that manipulating the BN layers could be effective for DA problems. Moreover, if combined with network pruning, pruning channels in a feature map is equivalent to setting corresponding scaling factors to 0. Compared with other direct channel pruning methods such as TCP [104], the implementation of pruning BN scaling factors is much easier. Therefore, in this work statistics in the BN layers are leveraged for channel pruning without introducing extra parameters.

Essentially, redundant channels are supposed to impact the least to the training loss compared with other important ones, based on which the model pruning can be formulated as an optimization problem. The objective is to minimize the loss change after removing a set of channels. For a network with the BN layers, removing a channel $\mathbf{X}_{i,j}$ (the $j$-th channel in the $i$-th layer) is equivalent to setting the corresponding BN scaling factor $\gamma_{i,j}$ as 0. To facilitate the analysis, let $\Gamma = \{\gamma_{i,j}\}$ denote the full set of the BN scaling factors in the network. Given a set of pruning candidates $\Gamma' \subset \Gamma$, let function $h(\Gamma')$ denote the loss change after setting $\gamma \in \Gamma'$ to 0. The objective is to find such a subset of $\Gamma$ such that the loss change is minimized.

$$\begin{aligned} \min_{\Gamma'} \quad & h(\Gamma') \\ \text{s.t.} \quad & \Gamma' \subset \Gamma, \\ & h(\Gamma') = \left| \mathcal{L}(\mathbf{W}, \Gamma' = \mathbf{0}) - \mathcal{L}(\mathbf{W}, \Gamma') \right|, \\ & \operatorname{card}(\Gamma') = P, \end{aligned} \tag{6.22}$$

where $\mathcal{L}(\mathbf{W}, \Gamma' = \mathbf{0})$ and $\mathcal{L}(\mathbf{W}, \Gamma')$ represent the corresponding loss for those channels are pruned and kept, respectively. $P$ is number of channels to be removed each time. $\operatorname{card}(\Gamma')$ denotes the total number of elements in set $\Gamma'$.

Solving this combinatorial problem exactly requires exhaustively evaluating all the possible combinations of $P$ channels in the network, which is not practical due to the intensive computation. Instead, we use a greedy methodology based on Taylor expansion for selection. A similar approach has been studied to prune individual parameters in the network [172]. We transform this approach to tackle channel pruning based on the BN scaling factors. In contrast to evaluating a subset $\Gamma'$ in the network, we evaluate each individual scaling factor first and rank them based on their impacts. Then a subset $\Gamma'$ is formed by selecting $P$ items with the least impacts. The Taylor expansion for an infinitely differentiable function $f(x)$ at point $x = a$ is represented as in Equation (6.23).

$$f(x) = \sum_{p=0}^{P} \frac{f^{(p)}(a)}{p!}(x-a)^p + R_p(x). \tag{6.23}$$

Therefore, the loss function $\mathcal{L}(\mathbf{W}, \gamma_{i,j})$ near $\gamma_{i,j} = \mathbf{0}$ can be approximated as Equation (6.24).

$$\mathcal{L}(\mathbf{W}, \gamma_{i,j} = 0) = \mathcal{L}(\mathbf{W}, \gamma_{i,j}) - \frac{\delta \mathcal{L}}{\delta \gamma_{i,j}} \gamma_{i,j} + R_1(\gamma_{i,j} = 0). \tag{6.24}$$

Here $R_1(\gamma_{i,j} = 0)$ is the Lagrange form remainder which is neglected due to the heavy computation required and marginal impacts on the results [172]. Then $h(\gamma_{i,j})$ can be approximated with

$$h(\gamma_{i,j}) = \left| \frac{\delta \mathcal{L}}{\delta \gamma_{i,j}} \gamma_{i,j} \right|. \tag{6.25}$$

The first term can be derived in backward computation and the second term is the current value of the scaling factor, thus Equation (6.25) can be computed efficiently.

### 6.2.3 Overall Training Steps

With the introduced training loss and pruning criterion, the training and compression of the partial domain adaptation model can be seamlessly integrated into a single-stage process. Pruning and model training are iteratively performed. Specifically, pruning is performed for every $T$ epochs of model training, where $T$ is user-defined. The stop condition is set to be the desired trade-off between the performance on the target domain and the execution overhead.

Table 6.7: The performance comparison on Office-31 dataset with VGG-16 as the base network.

| Tasks | VGG [150] | DAN [93] | WDAN [162] | SAN [100] | ETN [102] | TCP [104] | | | Ours-Pr. | Ours | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc. | Acc. | Acc. | Acc. | Acc. | Acc. | Param. | FLOPs | Acc. | Acc. | Param. | FLOPs |
| A31-W10 | 60.34 | 58.78 | 71.52 | 83.39 | 85.66 | 52.88 | -65% | -57% | 83.27 | 85.08 | -63% | -88% |
| A31-D10 | 76.34 | 54.76 | 73.88 | 90.70 | 89.43 | 46.50 | -63% | -60% | 94.27 | 91.08 | -61% | -87% |
| W31-A10 | 79.12 | 67.29 | 92.28 | 91.85 | 92.28 | 54.18 | -56% | -59% | 92.28 | 92.28 | -62% | -76% |
| W31-D10 | 99.36 | 92.78 | 96.17 | 100.00 | 100.00 | 91.00 | -58% | -61% | 99.36 | 99.36 | -68% | -84% |
| D31-A10 | 72.96 | 55.42 | 71.18 | 87.16 | 95.93 | 50.73 | -63% | -51% | 94.98 | 91.85 | -57% | -64% |
| D31-W10 | 97.97 | 85.86 | 87.45 | 99.32 | 100.00 | 84.41 | -61% | -64% | 100.00 | 99.32 | -58% | -70% |
| Average | 81.03 | 69.15 | 82.08 | 92.07 | 93.88 | 63.28 | -61% | -59% | 94.02 | 93.16 | -62% | -78% |

Table 6.8: The performance comparison on Office-31 dataset with ResNet-50 as the base network

| Tasks | ResNet [173] | DAN [93] | WDAN [162] | SAN [100] | ETN [102] | TCP [104] | | | Ours-Pr. | Ours | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc. | Acc. | Acc. | Acc. | Acc. | Acc. | Param. | FLOPs | Acc. | Acc. | Param. | FLOPs |
| A31-W10 | 75.59 | 59.32 | 73.55 | 93.90 | 94.52 | 48.81 | -59% | -50% | 95.25 | 94.58 | -70% | -80% |
| A31-D10 | 83.44 | 61.78 | 78.17 | 94.27 | 95.03 | 60.50 | -49% | -40% | 94.27 | 91.72 | -63% | -50% |
| W31-A10 | 84.97 | 67.64 | 93.52 | 88.73 | 94.64 | 56.57 | -58% | -50% | 94.89 | 94.05 | -73% | -60% |
| W31-D10 | 98.09 | 90.45 | 98.08 | 99.36 | 100.00 | 91.71 | -67% | -60% | 99.36 | 99.36 | -80% | -70% |
| D31-A10 | 89.92 | 74.95 | 92.17 | 94.15 | 96.21 | 55.32 | -57% | -50% | 95.30 | 94.08 | -63% | -50% |
| D31-W10 | 96.27 | 73.90 | 87.11 | 99.32 | 100.00 | 78.64 | -67% | -60% | 99.32 | 98.64 | -71% | -60% |
| Average | 87.05 | 71.34 | 87.10 | 94.96 | 96.73 | 65.26 | -60% | -52% | 96.40 | 95.41 | -70% | -62% |

### 6.2.4 Experimental Results

We conduct experiments on two benchmark datasets to evaluate the efficacy of the proposed approach. We compare the results with several other deep learning-based domain adaptation methods.

#### 6.2.4.1 Setup

Two public datasets are adopted in our experiments, including Office-31 and ImageCLEF-DA. Office-31 is a widely used benchmark for domain adaptation. There are in total of 31 categories and 4652 images in this dataset. The images are divided into three distinct domains which are denoted by Amazon (`A`), Webcam (`W`) and DSLR (`D`). There are 10 categories that are shared between Caltech-256 and Office-31. Then we use these shared 10 categories only in each domain of Office-31 as target domain, thus we can build six partial domain adaptation tasks: `A31-D10`, `A31-W10`, `W31-D10`, `W31-A10`, `D31-W10`, `D31-A10`.

ImageCLEF-DA is a benchmark for the ImageCLEF 2014 domain adaptation challenge. It contains four domains which are formed by selecting images from four public datasets, including Caltech-256 (`C`), ImageNet ILSVRC 2012 (`I`), Pascal VOC 2012 (`P`) and Bing (`B`). Each domain consists of 12 categories and each category contains 50 images. To perform partial domain adaptation, only the first 6 classes are selected and the remaining classes are discarded in the target domain. Following [104], we build 6 partial domain adaptation tasks: `I12-P6`, `P12-I6`, `I12-C6`, `C12-I6`, `P12-C6`, `C12-P6`.

We compare the performance of the proposed approach with other state-of-the-art unsupervised domain adaptation and network compression methods, including fine-tuned CNN, Deep Adaptation Network (**DAN**) [93], Weighted Domain Adaptation Network (**WDAN**) [162], Selective Adversarial Networks (**SAN**) [100], Example Transfer Networks (**ETN**) [102] and Transfer Channel Pruning (**TCP**) [104]. SAN targets at the partial domain adaptation task, which eases negative transfer by ruling out the outlier source classes and promotes positive transfer by maximally matching the data distributions in the shared label space using multiple branch discriminators. Note that Importance Weighted GAN (**IWGAN**) [101] is also a method for PDA. ETN [102] has made a comparison with [101] on the same tasks, in which [101] is dominated by ETN or SAN. So we only list SAN and ETN as baselines for comparison. TCP is designed for pruning less important channels while simultaneously learning transferable features by reducing the cross-domain distribution divergence. For a fair comparison, the experiments are conducted using two different base-networks, including VGG-16 [150] and

ResNet-50 [173], which are also used in SAN, TCP, and ETN.

We implement all the approaches based on `PyTorch` deep learning framework. The training starts from VGG-16 and ResNet-50 model pre-trained on ImageNet, which are provided by PyTorch. The soft weighted MMD layer is added before the last fully connected layer. We use mini-batch stochastic gradient descent (SGD) with the momentum of 0.9 and the weight decay of $5 \times 10^{-4}$. The learning rate is dynamically adjusted during the training process using the rule applied in [97, 100]: $lr = \frac{lr_0}{(1+\alpha p)^{\tau}}$, where $lr_0 = 10^{-3}$, $\alpha = 10$ and $\tau = 0.75$. $p$ is linearly adjusted from 0 to 1 during the training process. The user-defined penalty weights of soft weighted MMD loss and entropy loss are gradually increasing during the training. A similar way is applied in [104]. Since the model training should focus on the source data first, the cross-entropy loss should be relatively large. As the knowledge is retrieved during the optimization, the model should switch to focus on the target dataset, and the SWMMD loss and the entropy loss should have larger weights. The update rule is set as $2/(1 + e^{-it/IT}) - 1$, where $IT$ is the total number of training iterations and $it \in [0, IT]$ is the current iteration. The number of channels pruned $P$ is set to 128. The specific hyper-parameters are selected through cross-validation.

#### 6.2.4.2 Results

In our experiments, three metrics are leveraged to evaluate different methods on the partial domain adaptation tasks, including classification accuracy on the target domain, model size and total floating-point operations (FLOPs) of a complete inference. The FLOPs in a convolutional layer is calculated as $2HW(C_{in}K^2 + 1)C_{out}$, where $H$, $W$ and $C_{in}$ are height, width and number of channels of the input feature map. $K$ is the kernel

Table 6.9: The performance on ImageCLEF-DA with VGG-16 as base network.

| Tasks | VGG [150] | DAN [93] | TCP [104] | | | Ours-Pr. | Ours | | |
|---|---|---|---|---|---|---|---|---|---|
| | Acc. | Acc. | Acc. | Param. | FLOPs | Acc. | Acc. | Param. | FLOPs |
| P12-C6 | 94.00 | 92.67 | 63.67 | -40% | -50% | 94.00 | 96.00 | -46% | -60% |
| C12-P6 | 77.67 | 74.00 | 54.33 | -36% | -50% | 86.33 | 86.67 | -45% | -60% |
| P12-I6 | 88.67 | 85.67 | 61.33 | -37% | -50% | 88.00 | 88.67 | -49% | -60% |
| I12-P6 | 88.00 | 83.00 | 60.33 | -38% | -50% | 88.00 | 89.00 | -47% | -60% |
| C12-I6 | 82.33 | 82.00 | 56.00 | -40% | -50% | 90.00 | 87.33 | -45% | -60% |
| I12-C6 | 96.00 | 94.67 | 73.67 | -35% | -50% | 98.33 | 98.00 | -42% | -60% |
| Average | 87.78 | 73.17 | 61.56 | -38% | -50% | 90.78 | 90.95 | -46% | -60% |

Table 6.10: Performance on ImageCLEF-DA with ResNet-50 as base network.

| Tasks | ResNet [173] | DAN [93] | TCP [104] | | | Ours-Pr. | Ours | | |
|---|---|---|---|---|---|---|---|---|---|
| | Acc. | Acc. | Acc. | Param. | FLOPs | Acc. | Acc. | Param. | FLOPs |
| P12-C6 | 94.67 | 91.67 | 65.33 | -58% | -50% | 98.00 | 97.00 | -72% | -60% |
| C12-P6 | 79.00 | 73.67 | 53.00 | -58% | -51% | 88.33 | 83.33 | -73% | -60% |
| P12-I6 | 89.33 | 87.00 | 63.67 | -53% | -51% | 92.00 | 92.00 | -71% | -60% |
| I12-P6 | 89.67 | 85.67 | 62.57 | -57% | -51% | 87.00 | 86.67 | -73% | -60% |
| C12-I6 | 86.00 | 83.67 | 57.33 | -58% | -50% | 93.00 | 89.33 | -73% | -60% |
| I12-C6 | 96.00 | 94.00 | 77.00 | -58% | -50% | 97.00 | 97.00 | -73% | -60% |
| Average | 89.11 | 74.11 | 63.15 | -57% | -51% | 92.56 | 90.89 | -73% | -60% |

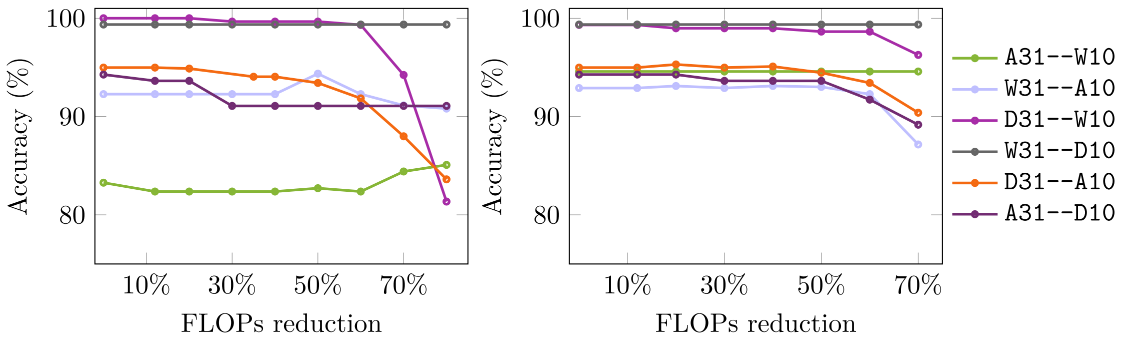


Figure 6.7: Accuracy on target domain vs. FLOPs reduction on Office-31 datasets. **Left**: VGG-16 as base network; **Right**: ResNet-50 as base network.

width and height, and $C_{out}$ is the number of channels of the output feature map. For a fully connected layer, we compute FLOPs as $(2I-1)O$, where $I$ and $O$ are the input dimensionality and the output dimensionality of a fully connected layer, respectively.

Firstly, we compare the performance of our method with WDAN, SAN, ETN and

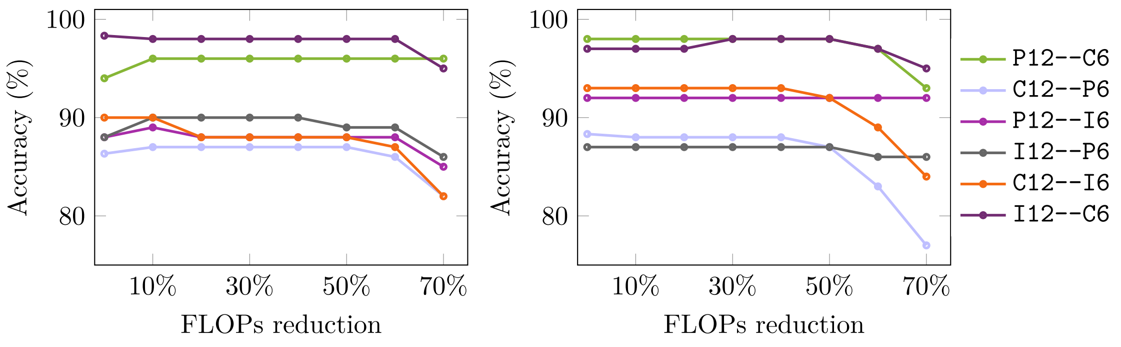


Figure 6.8: Accuracy on target domain vs. FLOPs reduction on ImageCLEF-DA datasets. **Left**: VGG-16 as base network; **Right**: ResNet-50 as base network.

TCP on the six partial domain adaptation tasks of the Office-31 dataset, which is presented in Table 6.7 and Table 6.8. We use the model size and FLOPs of original base networks as baselines and demonstrate the reduction in the model size and computation. SAN and ETN are based on adversarial training for PDA. The negative transfer can be circumvented well. However, once the network is trained, the model size computation overhead is the same as the original base network (i.e., VGG-16 or ResNet-50), which is very large. TCP performs similar strategies to iteratively prune channels and fine-tune. Since only fully shared label space between domains is considered, the negative transfer issue is not well addressed, thus leads to significant accuracy degradation when models become smaller. With the proposed SWMMD and pruning methods, all three aspects are taken good care of. For comparison, we also disable the pruning process and training the network for PDA and the performance is listed in "Ours-Pr.". Nearly 80% of FLOPs and more than 62% of parameters are reduced with less than 1% accuracy degradation compared to ETN. Notably, the accuracy results are shown in Table 6.7 achieve the same level or even outperform ETN on certain tasks such as A31-D10 and W31-A10, which validates that the redundancy does exist and removing that can be

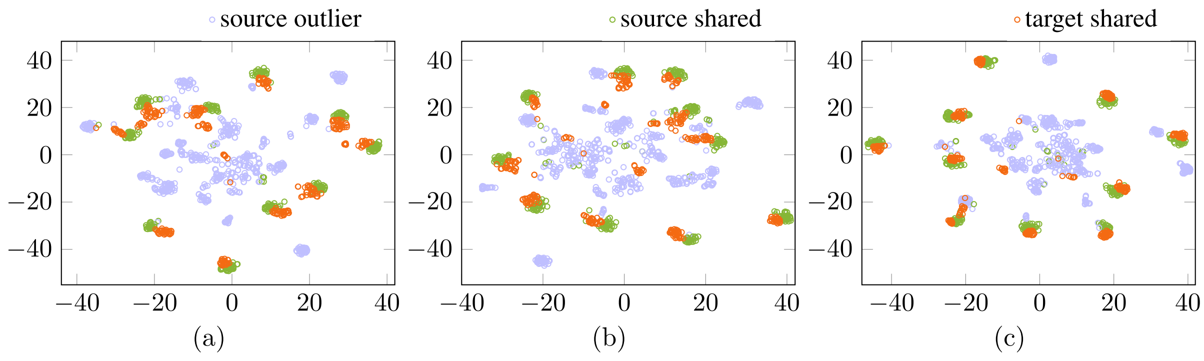


Figure 6.9: t-SNE visualization on learned features on task `A31-W10`. (a) ResNet-50; (b) DAN with ResNet-50; (c) Ours.

beneficial to PDA. Also, when ResNet-50 serves as the base network, 70% of parameters and 62% of FLOPs are reduced with only 1.3% loss on performance, which indicates that the proposed approach can generalize well to different CNN architectures.

Another experiment is conducted using the ImageCLEF-DA dataset. Following [104], six experiments are performed. Similar to the experiments on Office-31, both VGG-16 and ResNet-50 are used as base networks to compare different methods. DAN [93] ignores the change of the class prior distribution, and hence it will lead to negative transfer in PDA settings. Therefore, using base networks directly to perform domain adaptation, i.e., train on the source domain and test on the target domain, can achieve higher accuracy than DAN. If we do not apply pruning process and only train the base network with the SWMMD scheme, it can be observed from Table 6.9 and Table 6.10 that the accuracy can be improved by a large margin, which validates the effectiveness of the SWMMD method on negative transfer alleviation. If further combined with the pruning process, the redundancy is proved to exist and can be removed substantially. When the FLOPs are reduced by 60%, the number of parameters can be reduced by nearly 50% on VGG-16 and more than 70% on ResNet-50.

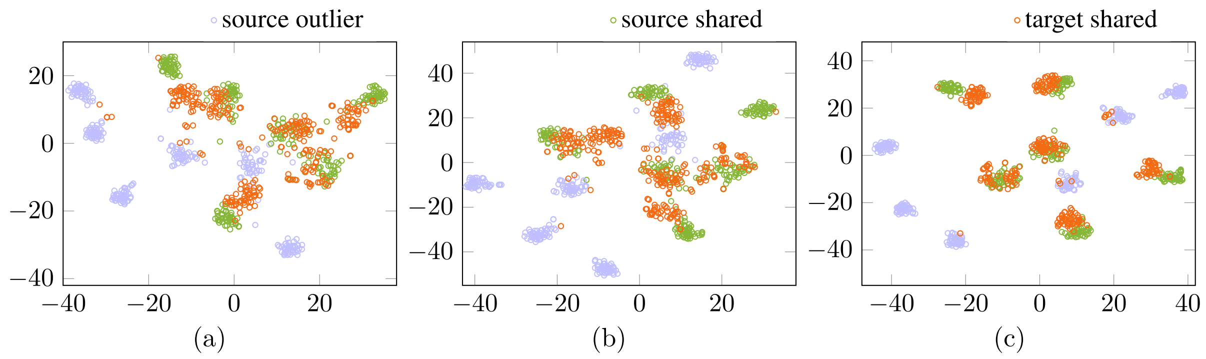


Figure 6.10: t-SNE visualization on learned features on task `C12-P6`. (a) ResNet-50; (b) DAN with ResNet-50; (c) Ours.

It can be observed from Table 6.9 that higher accuracy can be attained on VGG-16 when the pruning process is enabled, even though the pruning ratio is as large as 60%, which reveals that there exists huge redundancy in VGG-16. For ResNet-50, the accuracy slightly degrades when pruning ratio is 60%, as shown in Table 6.10, which indicates the redundancy issue is less severe than VGG-16 but still exists. The trade-off between overhead and accuracy is detailed analyzed in Section 6.2.4.3.

#### 6.2.4.3 Analysis

Intuitively, small size and high performance are naturally in conflict with each other to some extent. Only when there exists large redundancy can we find that smaller size can boost the performance. To analyze whether the assumed intensive redundancy exists in the PDA scenario, the trade-off curves can provide us the insight into this problem. The relation curves on Office-31 dataset are shown in Figure 6.7. With VGG-16 as the base network, it is observed that three tasks (D31-A10, D31-W10, A31-D10) reflect an obvious trade-off between accuracy and computation. The accuracy of the W31-D10 task keeps steady all the time until FLOPs are reduced by 80%, which validates the existence of large redundancy. On A31-W10 and W31-A10 tasks, there is even an increase in the accuracy as models get smaller. These observations reflect that transferring a large pre-trained model to a small scale task is unnecessary, and slimming a model can be beneficial to both performance and computation efficiency. With ResNet-50 as base network, noticeable trade-offs are observed in 4 out of 6 tasks. Similarly, the accuracy of A31-W10 and W31-D10 tasks are not impacted by the model size. While on ImageCLEF-DA, the trade-offs are reflected in nearly all the tasks on both bottleneck networks. Despite the performance degradation on the end, which may be because pruning is already very aggressive, all the curves shown in Figure 6.7 and Figure 6.8 keep steady for a while, indicating the room and the necessity of performing model pruning.

We visualize the learned representations to demonstrate the effectiveness of different methods with the t-SNE method [132] which embeds the representation of a high dimensional space into a 2-dimensional space. The representations learned in two tasks are presented, including `A31-W10` on Office-31 and `C12-P6` on ImageCLEF-DA, as shown in Figure 6.9 and Figure 6.10. Orange dots and green dots represent the shared classes between source and target domains. Blue dots represent outlier classes. Strong alignment between the orange dots and green dots indicate the effective circumvention of negative transfer. It can be observed that Figure 6.9a and Figure 6.10a show better alignment than Figure 6.9b and Figure 6.10b, which is expected. Figure 6.9c and Figure 6.10c show the strongest alignment, which suggests the advantage of the proposed method.

### 6.2.5 Summary

This work presents a new perspective on the challenging partial domain adaptation problem by integrating with neural network compression. An SWMMD is applied to match the cross-domain distribution when the label spaces are not identical in the source and the target domains. On top of that, a channel pruning method is developed to iteratively prune channels based on the corresponding scaling factors in the BN layer. Experimental results indicate the proposed approach can simultaneously achieve compelling accuracy, smaller model size and fewer computations compared to other model pruning and domain adaptation works.

# Chapter 7

# Conclusion

In this thesis, a set of methodologies are proposed for improving the efficiency of the hardware design automation and enabling hardware-friendly learning, covering several significant stages in typical design flow.

- In Chapter 3, we enhance a state-of-the-art prefix adder synthesis algorithm to obtain a much wider solution space in architectural domain. On top of that, a machine learning-based design space exploration methodology is applied to predict the Pareto frontier of the adders in physical domain. Considering the high cost of obtaining the true values for learning, an active learning algorithm is proposed to use less labeled data while achieving better quality of Pareto frontier. Experimental results demonstrate that our framework can achieve Pareto frontier of high quality over a wide design space, bridging the gap between architectural and physical designs.

- In Chapter 4, a high performance graph convolutional network model is proposed for the purpose of processing irregular graph representations of logic circuits. A

GCN classifier is firstly trained to predict observation point candidates in a netlist, and is used as part of an iterative process for observation point insertion. Experimental results show the proposed GCN model has superior accuracy to classical machine learning models on difficult-to-observation nodes prediction. Compared with commercial testability analysis tools, the proposed observation point insertion flow achieves similar fault coverage with an 11% reduction in observation points and a 6% reduction in test pattern count.

- In Chapter 5, we propose a unified framework LDMO, which seamlessly integrates layout decomposition and mask optimization. We propose a gradient-based approach to solve the unified mathematical formulation, as well as a set of discrete optimization techniques to avoid being stuck in local optimum. The conventional optimization process can be accelerated as some inferior decomposition results can be smartly pruned in early stages. The experimental results show that the proposed unified framework can achieve more than 34$\times$ speed-up compared with the conventional two-stage flow, meanwhile it can dramatically reduce EPE violations by more than 8$\times$, and thus maintain better design quality.

- In Chapter 6, we propose different methodologies for hardware-friendly learning in different scenarios. Firstly, in Section 6.1, we propose a unified framework to compress the CNNs by combining the low-rankness and sparsity. Each layer in the network is approximated by the sum of a structured sparse component and a low-rank component, which is formulated as an optimization problem. Then, an extended version of ADMM with guaranteed convergence is presented to solve the relaxed optimization problem. Experiments carried out on conventional networks with large image classification datasets show that the proposed method is able to remarkably compress the model (with up to 4.9× reduction of parameters) at a cost of little loss or without loss on accuracy, and outperform previous work in terms of accuracy degradation, compression rate and speedup ratio. In Section 6.2, partial domain adaptation and model compression are seamlessly integrated into a unified training process. The cross domain distribution divergence is reduced by minimizing a soft-weighted maximum mean discrepancy. To compress the over-parameterized model, we utilize the gradient statistics to identify and prune redundant channels based on the corresponding scaling factors in batch normalization layers. Experimental results demonstrate that our method can achieve comparable classification performance to the state-of-the-art methods on various partial domain adaptation tasks, with significant reduction on model size and computation overhead.

Although a set of new methodologies have been investigated in this thesis, there are still enormous challenges in the design automation and system integration. With the continuously increasing difficulty in design, manufacturing and integration, it is highly expected that more and more methodologies could arise to further push forward the technology innovation. Particularly, the following research problems and directions would be worthy exploring.

- The mainstream learning-based methodologies in design automation are supervised learning, which requires much expertise to obtain well-designed features for the objects, e.g., the SCOAP values in our DFT problem (Chapter 4), $mfo$ and $mpfo$ in DSE problem (Chapter 3). Although deep learning has eased the problem of purely manual feature extraction, suitable representation or pre-processing is still essential to the performance, as demonstrated in layout verification and mask synthesis problems [174, 175, 176, 75]. Therefore, unsupervised learning or feature learning for automatic embedding generation could be a promising technique to further boost the learning-based methodologies.

- There are many scenarios in design automation that could lead to data distribution discrepancy issue. For example, circuits may be designed under different technology nodes, or for totally different functional modules, which are referred as different domains. In these cases, training a universal model for data in all domains could not be practical. Transfer learning is a potential solution to bridge the distribution discrepancy, which could help to exploit the knowledge from one domain and apply to other domains without training a new model from scratch. It is also of great significance in terms of reducing the data labeling cost.

- Current learning-based methodologies mainly play as assistant modules in the design flow, which provide certain feedback to designers such that they can make decisions more efficiently and more accurately. In order to further improve the efficiency, reinforcement learning is another technique with great hopes invested. By training an agent for sequential decision making on designing hardware and system, it is expected that the target could get improved gradually. It would achieve a "no-human-in-the-loop" diagram and shorten the design cycle dramatically.

# Bibliography

[1] Subhendu Roy, Mihir Choudhury, Ruchir Puri, and David Z. Pan. Towards optimal performance-area trade-off in adders by synthesis of parallel prefix structures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 33(10):1517–1530, 2014.

[2] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Conference on Neural Information Processing Systems (NIPS)*, pages 1024–1034, 2017.

[3] N. H. Weste and D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison Wesley, 2004.

[4] Bei Yu, Yen-Hung Lin, Gerard Luk-Pat, Duo Ding, Kevin Lucas, and David Z. Pan. A high-performance triple patterning layout decomposer with balanced density. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 163–169, 2013.

[5] Jhih-Rong Gao, Xiaoqing Xu, Bei Yu, and David Z. Pan. MOSAIC: Mask optimizing solution with process window aware inverse correction. In *ACM/IEEE Design Automation Conference (DAC)*, pages 52:1–52:6, 2014.

[6] Zihao Chen, Hailong Yao, and Yici Cai. SUALD: Spacing uniformity-aware layout decomposition in triple patterning lithography. In *IEEE International Symposium on Quality Electronic Design (ISQED)*, pages 566–571, 2013.

[7] Yuzhe Ma, Jhih-Rong Gao, Jian Kuang, Jin Miao, and Bei Yu. A unified framework for simultaneous layout decomposition and mask optimization. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 81–88, 2017.

[8] Subhendu Roy, Yuzhe Ma, Jin Miao, and Bei Yu. A learning bridge from architectural synthesis to physical design for exploring power efficient high-performance adders. In *IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2017.

[9] Hung-Yi Liu and Luca P. Carloni. On learning-based methods for design-space exploration with high-level synthesis. In *ACM/IEEE Design Automation Conference (DAC)*, pages 50:1–50:7, 2013.

[10] Pingfan Meng, Alric Althoff, Quentin Gautier, and Ryan Kastner. Adaptive threshold non-pareto elimination: Re-thinking machine learning for system level design space exploration on FPGAs. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, pages 918–923, 2016.

[11] Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. ReSPIR: a response surface-based pareto iterative refinement for application-specific design space exploration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 28(12):1816–1829, 2009.

[12] Zhiyao Xie, Guan-Qi Fang, Yu-Hung Huang, Haoxing Ren, Yanqing Zhang, Brucek Khailany, Shao-Yun Fang, Jiang Hu, Yiran Chen, and Erick Carvajal Barboza. Fist: A feature-importance sampling and tree-based method for automatic design flow parameter tuning. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 19–25, 2020.

[13] Matthew M Ziegler, Hung-Yi Liu, George Gristede, Bruce Owens, Ricardo Nigaglioni, and Luca P Carloni. A synthesis-parameter tuning system for autonomous design-space exploration. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, pages 1148–1151, 2016.

[14] Jihye Kwon, Matthew M Ziegler, and Luca P Carloni. A learning-based recommender system for autotuning design fiows of industrial high-performance processors. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.

[15] Matthew M Ziegler, Hung-Yi Liu, and Luca P Carloni. Scalable auto-tuning of synthesis parameters for optimizing high-performance processors. In *IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pages 180–185, 2016.

[16] Wenlong Lyu, Fan Yang, Changhao Yan, Dian Zhou, and Xuan Zeng. Multi-objective bayesian optimization for analog/rf circuit synthesis. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2018.

[17] Wenlong Lyu, Fan Yang, Changhao Yan, Dian Zhou, and Xuan Zeng. Batch bayesian optimization via multi-objective acquisition ensemble for automated analog circuit design. In *International Conference on Machine Learning (ICML)*, pages 3306–3314, 2018.

[18] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Process for Machine Learning*. The MIT Press, 2006.

[19] Shuhan Zhang, Wenlong Lyu, Fan Yang, Changhao Yan, Dian Zhou, Xuan Zeng, and Xiangdong Hu. An efficient multi-fidelity bayesian optimization approach for analog circuit synthesis. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.

[20] Shuhan Zhang, Wenlong Lyu, Fan Yang, Changhao Yan, Dian Zhou, and Xuan Zeng. Bayesian optimization approach for analog circuit synthesis using neural network. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, pages 1463–1468, 2019.

[21] David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In *ACM SIGIR Conference*, pages 3–12, 1994.

[22] Michael Lindenbaum, Shaul Markovitch, and Dmitry Rusakov. Selective sampling for nearest neighbor classifiers. *Machine learning*, 54(2):125–152, 2004.

[23] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, 2(Nov.):45–66, 2001.

[24] Edoardo Pasolli, Farid Melgani, Devis Tuia, Fabio Pacifici, and William J. Emery. SVM active learning approach for image classification using spatial information. *IEEE Transactions on Geoscience and Remote Sensing (TGRS)*, 52(4):2217–2233, 2014.

[25] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. *Journal of artificial intelligence research*, 4:129–145, 1996.

[26] Patrick Flaherty, Adam Arkin, and Michael I Jordan. Robust design of biological experiments. In *Conference on Neural Information Processing Systems (NIPS)*, pages 363–370, 2006.

[27] Guillaume Sergent Marcela Zuluaga, Andreas Krause and Markus P*ü*schel. Active learning for multi-objective optimization. In *International Conference on Machine Learning (ICML)*, pages 462–470, 2013.

[28] H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294, 1992.

[29] Naoki Abe Hiroshi Mamitsuka et al. Query learning strategies using boosting and bagging. In *International Conference on Machine Learning (ICML)*, volume 1, 1998.

[30] Prem Melville and Raymond J Mooney. Diverse ensembles for active learning. In *International Conference on Machine Learning (ICML)*, page 74, 2004.

[31] Andrew Kachites McCallumzy and Kamal Nigamy. Employing em and pool-based active learning for text classification. In *International Conference on Machine Learning (ICML)*, pages 359–367, 1998.

[32] Burr Settles, Mark Craven, and Soumya Ray. Multiple-instance active learning. In *Conference on Neural Information Processing Systems (NIPS)*, pages 1289–1296, 2008.

[33] Burr Settles and Mark Craven. An analysis of active learning strategies for sequence labeling tasks. In *Empirical Methods in Natural Language Processing*, pages 1070–1079, 2008.

[34] Stanley F Chen and Ronald Rosenfeld. A survey of smoothing techniques for me models. *IEEE Transactions on Speech and Audio Processing*, 8(1):37–50, 2000.

[35] Balakrishnan Krishnamurthy. A dynamic programming approach to the test point insertion problem. In *ACM/IEEE Design Automation Conference (DAC)*, pages 695–705, 1987.

[36] A. J. Briers and K. A. E. Totton. Random pattern testability by fast fault simulation. In *IEEE International Test Conference (ITC)*, 1986.

[37] MJ Geuzebroek, J Th Van Der Linden, and AJ Van de Goor. Test point insertion that facilitates atpg in reducing test time and data volume. In *IEEE International Test Conference (ITC)*, pages 138–147, 2002.

[38] Nagesh Tamarapalli and Janusz Rajski. Constructive multi-phase test point insertion for scan-based bist. In *IEEE International Test Conference (ITC)*, pages 649–658, 1996.

[39] Joon-Sung Yang, Nur A Touba, Benoit Nadeau-Dostie, et al. Test point insertion with control points driven by existing functional flip-flops. *IEEE Transactions on Computers*, 61(10):1473–1483, 2012.

[40] Nur A Touba and Edward J McCluskey. Test point insertion based on path tracing. In *IEEE VLSI Test Symposium (VTS)*, pages 2–8, 1996.

[41] Haoxing Ren, Mary Kusko, Victor Kravets, and Rona Yaari. Low cost test point insertion without using extra registers for high performance design. In *IEEE International Test Conference (ITC)*, pages 1–8, 2009.

[42] Hon Fung Li and PN Lam. A protocol extraction strategy for control point insertion in design for test of transition signaling circuits. In *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, pages 178–183, 1995.

[43] Zipeng Li, Sandeep Kumar Goel, Frank Lee, and Krishnendu Chakrabarty. Efficient observation-point insertion for diagnosability enhancement in digital circuits. In *IEEE International Test Conference (ITC)*, pages 1–10, 2015.

[44] Hongyun Cai, Vincent W Zheng, and Kevin Chang. A comprehensive survey of graph embedding: problems, techniques and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[45] Smriti Bhagat, Graham Cormode, and S Muthukrishnan. Node classification in social networks. In *Social network data analytics*, pages 115–148. Springer, 2011.

[46] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 855–864, 2016.

[47] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 974–983, 2018.

[48] Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. In *International Conference on Learning Representations (ICLR)*, 2020.

[49] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, et al. Deep graph library: Towards efficient and scalable deep learning on graphs. *arXiv preprint arXiv:1909.01315*, 2019.

[50] Andrew B. Kahng, Chul-Hong Park, Xu Xu, and Hailong Yao. Layout decomposition approaches for double patterning lithography. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 29:939–952, June 2010.

[51] Yue Xu and Chris Chu. GREMA: graph reduction based efficient mask assignment for double patterning technology. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 601–606, 2009.

[52] Kun Yuan, Jae-Seok Yang, and David Z. Pan. Double patterning layout decomposition for simultaneous conflict and stitch minimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 29(2):185–196, Feb. 2010.

[53] Bei Yu, Kun Yuan, Duo Ding, and David Z. Pan. Layout decomposition for triple patterning lithography. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 34(3):433–446, March 2015.

[54] Bei Yu, Subhendu Roy, Jhih-Rong Gao, and David Z. Pan. Triple patterning lithography layout decomposition using end-cutting. *Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3)*, 14(1):011002–011002, 2015.

[55] Yibo Lin, Xiaoqing Xu, Bei Yu, Ross Baldick, and David Z. Pan. Triple/quadruple patterning layout decomposition via novel linear programming and iterative rounding. In *SPIE Advanced Lithography*, volume 9781, 2016.

[56] Xingquan Li, Ziran Zhu, and Wenxing Zhu. Discrete relaxation method for triple patterning lithography layout decomposition. *IEEE Transactions on Computers*, 66(2):285–298, 2017.

[57] Shao-Yun Fang, Yao-Wen Chang, and Wei-Yu Chen. A novel layout decomposition algorithm for triple patterning lithography. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 33(3):397–408, March 2014.

[58] Hsi-An Chien, Szu-Yuan Han, Ye-Hong Chen, and Ting-Chi Wang. A cell-based row-structure layout decomposer for triple patterning lithography. In *ACM International Symposium on Physical Design (ISPD)*, pages 67–74, 2015.

[59] Haitong Tian, Hongbo Zhang, Qiang Ma, Zigang Xiao, and Martin D. F. Wong. A polynomial time triple patterning algorithm for cell based row-structure layout. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 57–64, 2012.

[60] Jian Kuang and Evangeline F. Y. Young. Fixed-parameter tractable algorithms for optimal layout decomposition and beyond. In *ACM/IEEE Design Automation Conference (DAC)*, pages 61:1–61:6, 2017.

[61] Jian Kuang and Evangeline F. Y. Young. An efficient layout decomposition approach for triple patterning lithography. In *ACM/IEEE Design Automation Conference (DAC)*, pages 69:1–69:6, 2013.

[62] Ye Zhang, Wai-Shing Luk, Hai Zhou, Changhao Yan, and Xuan Zeng. Layout decomposition with pairwise coloring for multiple patterning lithography. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 170–177, 2013.

[63] Hua-Yu Chang and Iris Hui-Ru Jiang. Multiple patterning layout decomposition considering complex coloring rules. In *ACM/IEEE Design Automation Conference (DAC)*, pages 40:1–40:6, 2016.

[64] Weiping Fang, Srini Arikati, Erdem Cilingir, Marco A Hug, Peter De Bisschop, Julien Mailfert, Kevin Lucas, and Weimin Gao. A fast triple-patterning solution with fix guidance. In *Proceedings of SPIE*, volume 9053, 2014.

[65] Ahmed Awad, Atsushi Takahashi, Satoshi Tanaka, and Chikaaki Kodama. A fast process variation and pattern fidelity aware mask optimization algorithm. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 238–245, 2014.

[66] Jian Kuang, Wing-Kai Chow, and Evangeline F. Y. Young. A robust approach for process variation aware mask optimization. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, pages 1591–1594, 2015.

[67] Yu-Hsuan Su, Yu-Chen Huang, Liang-Chun Tsai, Yao-Wen Chang, and Shayak Banerjee. Fast lithographic mask optimization considering process variation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 35(8):1345–1357, 2016.

[68] Amyn Poonawala and Peyman Milanfar. Mask design for optical microlithography–an inverse imaging problem. *IEEE Transactions on Image Processing*, 16(3):774–788, 2007.

[69] Ningning Jia and Edmund Y. Lam. Machine learning for inverse lithography: using stochastic gradient descent for robust photomask synthesis. *Journal of Optics*, 12(4):045601:1–045601:9, 2010.

[70] Yijiang Shen, Ngai Wong, and Edmund Y. Lam. Level-set-based inverse lithography for photomask synthesis. *Optics Express*, 17(26):23690–23701, Dec 2009.

[71] Wen Lv, Shiyuan Liu, Qi Xia, Xiaofei Wu, Yijiang Shen, and Edmund Y Lam. Level-set-based inverse lithography for mask synthesis using the conjugate gradient and an optimal time step. *Journal of Vacuum Science & Technology B, Nanotechnology and Microelectronics: Materials, Processing, Measurement, and Phenomena*, 31(4):041605, 2013.

[72] Rui Luo. Optical proximity correction using a multilayer perceptron neural network. *Journal of Optics*, 15(7):075708, 2013.

[73] Allan Gu and Avideh Zakhor. Optical proximity correction with linear regression. *IEEE Transactions on Semiconductor Manufacturing (TSM)*, 21(2):263–271, 2008.

[74] Tetsuaki Matsunawa, Bei Yu, and David Z. Pan. Optical proximity correction with hierarchical bayes model. In *Proceedings of SPIE*, volume 9426, 2015.

[75] Haoyu Yang, Shuhe Li, Yuzhe Ma, Bei Yu, and Evangeline F.Y. Young. GAN-OPC: Mask optimization with lithography-guided generative adversarial nets. In *ACM/IEEE Design Automation Conference (DAC)*, pages 131:1–131:6, 2018.

[76] Misha Denil, Babak Shakibi, Laurent Dinh, Nando De Freitas, et al. Predicting parameters in deep learning. In *Conference on Neural Information Processing Systems (NIPS)*, pages 2148–2156, 2013.

[77] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.

[78] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations (ICLR)*, 2017.

[79] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Conference on Neural Information Processing Systems (NIPS)*, pages 2074–2082, 2016.

[80] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[81] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. NISP: Pruning networks using neuron importance score propagation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[82] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Conference (BMVC)*, 2014.

[83] Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. Tensorizing neural networks. In *Conference on Neural Information Processing Systems (NIPS)*, pages 442–450, 2015.

[84] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned CP-decomposition. In *International Conference on Learning Representations (ICLR)*, 2015.

[85] Shiva Prasad Kasiviswanathan, Nina Narodytska, and Hongxia Jin. Network approximation using tensor sketching. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2319–2325, 2018.

[86] Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. Convolutional neural networks with low-rank regularization. In *International Conference on Learning Representations (ICLR)*, 2016.

[87] Jose M. Alvarez and Mathieu Salzmann. Compression-aware training of deep networks. In *Conference on Neural Information Processing Systems (NIPS)*, pages 856–867, 2017.

[88] Rui Dai, Lefei Li, and Wenjian Yu. Fast training and model compression of gated RNNs via singular value decomposition. In *International Joint Conference on Neural Networks (IJCNN)*, 2018.

[89] Wei Wen, Cong Xu, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Coordinating filters for faster deep neural networks. In *IEEE International Conference on Computer Vision (ICCV)*, pages 658–666, 2017.

[90] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2009.

[91] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks (TNN)*, 22(2):199–210, 2010.

[92] Boqing Gong, Kristen Grauman, and Fei Sha. Connecting the dots with landmarks: Discriminatively learning domain-invariant features for unsupervised domain adaptation. In *International Conference on Machine Learning (ICML)*, pages 222–230, 2013.

[93] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *International Conference on Machine Learning (ICML)*, pages 97–105, 2015.

[94] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Unsupervised domain adaptation with residual transfer networks. In *Conference on Neural Information Processing Systems (NIPS)*, pages 136–144, 2016.

[95] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. *arXiv preprint arXiv:1412.3474*, 2014.

[96] Werner Zellinger, Thomas Grubinger, Edwin Lughofer, Thomas Natschläger, and Susanne Saminger-Platz. Central moment discrepancy (cmd) for domain-invariant representation learning. *arXiv preprint arXiv:1702.08811*, 2017.

[97] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, pages 2096–2030, 2016.

[98] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7167–7176, 2017.

[99] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1–2):151–175, 2010.

[100] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Michael I Jordan. Partial transfer learning with selective adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2724–2732, 2018.

[101] Jing Zhang, Zewei Ding, Wanqing Li, and Philip Ogunbona. Importance weighted adversarial nets for partial domain adaptation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8157–8164, 2018.

[102] Zhangjie Cao, Kaichao You, Mingsheng Long, Jianmin Wang, and Qiang Yang. Learning to transfer examples for partial domain adaptation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2985–2994, 2019.

[103] Zhangjie Cao, Lijia Ma, Mingsheng Long, and Jianmin Wang. Partial adversarial domain adaptation. In *European Conference on Computer Vision (ECCV)*, pages 135–150, 2018.

[104] Chaohui Yu, Jindong Wang, Yiqiang Chen, and Zijing Wu. Accelerating deep unsupervised domain adaptation with transfer channel pruning. *arXiv preprint arXiv:1904.02654*, 2019.

[105] Bei Yu, David Z. Pan, Tetsuaki Matsunawa, and Xuan Zeng. Machine learning and pattern matching in physical design. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 286–293, 2015.

[106] Wei-Ting J. Chan, Kun Young Chung, Andrew B. Kahng, Nancy D. MacDonald, and Siddhartha Nath. Learning-based prediction of embedded memory timing failures during initial floorplan design. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 178–185, 2016.

[107] Wen-Hsiang Chang, Li-De Chen, Chien-Hsueh Lin, Szu-Pang Mu, Mango C-T. Chao, Cheng-Hong Tsai, and Yen-Chih Chiu. Generating routing-driven power distribution networks with machine-learning technique. In *ACM International Symposium on Physical Design (ISPD)*, pages 145–152, 2016.

[108] Rupak Samanta, Jiang Hu, and Peng Li. Discrete buffer and wire sizing for link-based non-tree clock networks. In *ACM International Symposium on Physical Design (ISPD)*, pages 175–181, 2008.

[109] R. P. Brent and H. T. Kung. A regular layout for parallel adders. *IEEE Transactions on Computers*, C-31(3):260–264, 1982.

[110] Peter M. Kogge and Harold S. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Transactions on Computers*, 100(8):786–793, 1973.

[111] T. Han and D. Carlson. Fast area-efficient VLSI adders. *IEEE Symposium on Computer Arithmetic (ARITH)*, pages 49–56, 1987.

[112] J. Sklansky. Conditional sum addition logic. *IRE Trans. on Electronic Computers*, EC-9(2):226–231, 1960.

[113] B. R. Zeydel, T. T. J. H. Kluter, and V. G. Oklobdzija. Efficient mapping of addition recurrence algorithms in CMOS. *IEEE Symposium on Computer Arithmetic (ARITH)*, pages 107–113, 2005.

[114] M. Ketter, D. M. Harris, A. Macrae, R. Glick, M. Ong, and J. Schauer. Implementation of 32-bit Ling and Jackson adders. *Asilomar Conference on Signals, Systems, and Computers*, pages 170–175, 2011.

[115] Eckart Zitzler, Dimo Brockhoff, and Lothar Thiele. The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration. In *Evolutionary multi-criterion optimization*, pages 862–876, 2007.

[116] Synopsys Design Compiler. `http://www.synopsys.com`.

[117] Synopsys IC Compiler. `http://www.synopsys.com`.

[118] Synopsys SAED Library. `http://www.synopsys.com/Community/UniversityProgram/Pages/32-28nm-generic-library.aspx`. Accessed 23-April-2016.

[119] Kagan Tumer and Joydeep Ghosh. Estimating the bayes error rate through classifier combining. In *IEEE International Conference on Pattern Recognition (ICPR)*, volume 2, pages 695–699, 1996.

[120] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: no regret and experimental design. In *International Conference on Machine Learning (ICML)*, pages 1015–1022, 2010.

[121] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(Oct.):2825–2830, 2011.

[122] Subhendu Roy, Mihir Choudhury, Ruchir Puri, and David Z. Pan. Polynomial time algorithm for area and power efficient adder synthesis in high-performance designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 35(5):820–831, 2016.

[123] T. McAuley, W. Koven, A. Carter, P. Ning, and D. M. Harris. Implementation of a 64-bit Jackson adders. *Asilomar Conference on Signals, Systems, and Computers*, pages 1149–1154, 2013.

[124] Subhendu Roy, Mihir Choudhury, Ruchir Puri, and David Z. Pan. Polynomial time algorithm for area and power efficient adder synthesis in high-performance designs. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 249–254, 2015.

[125] Navaratnasothie Selvakkumaran and George Karypis. Multiobjective hypergraph-partitioning algorithms for cut and maximum subdomain-degree minimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 25(3):504–517, 2006.

[126] Tsung-Hsien Lee and Ting-Chi Wang. Congestion-constrained layer assignment for via minimization in global routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 27(9):1643–1656, 2008.

[127] Kwan-Ting Cheng and Chih-Jen Lin. Timing-driven test point insertion for full-scan and partial-scan BIST. In *IEEE International Test Conference (ITC)*, pages 506–514, 1995.

[128] Winston Haaswijk, Edo Collins, Benoit Seguin, Mathias Soeken, Frédéric Kaplan, Sabine Süsstrunk, and Giovanni De Micheli. Deep learning for logic optimization algorithms. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, 2018.

[129] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations (ICLR)*, 2016.

[130] Lawrence H Goldstein and Evelyn L Thigpen. SCOAP: Sandia controllability/observability analysis program. In *ACM/IEEE Design Automation Conference (DAC)*, pages 190–196, 1980.

[131] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*, pages 807–814, 2010.

[132] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[133] David Z. Pan, Bei Yu, and J.-R. Gao. Design for manufacturing with emerging nanolithography. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 32(10):1453–1472, 2013.

[134] Bei Yu, Xiaoqing Xu, Subhendu Roy, Yibo Lin, Jiaojiao Ou, and David Z. Pan. Design for manufacturability and reliability in extreme-scaling VLSI. *Science China Information Sciences*, pages 1–23, 2016.

[135] Nicolas Bailey Cobb. *Fast optical and process proximity correction algorithms for integrated circuit manufacturing*. PhD thesis, University of California at Berkeley, 1998.

[136] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[137] Brian Borchers. CSDP, a C library for semidefinite programming. *Optimization Methods and Software*, 11:613–623, 1999.

[138] Shayak Banerjee, Zhuo Li, and Sani R. Nassif. ICCAD-2013 CAD contest in mask optimization and benchmark suite. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 271–274, 2013.

[139] NanGate FreePDK45 Generic Open Cell Library. `http://www.si2.org/openeda.si2.org/projects/nangatelib`, 2008.

[140] John Randall, Kurt G Ronse, Thomas Marschner, Anne-Marie Goethals, and Monique Ercken. Variable-threshold resist models for lithography simulation. In *Optical Microlithography XII*, volume 3679, pages 176–182, 1999.

[141] Bei Yu, Kun Yuan, Boyang Zhang, Duo Ding, and David Z. Pan. Layout decomposition for triple patterning lithography. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2011.

[142] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *ACM Symposium on FPGAs*, pages 161–170, 2015.

[143] Hoi-Jun Yoo, Seongwook Park, Kyeongryeol Bong, Dongjoo Shin, Jinmook Lee, and Sungpill Choi. A 1.93 tops/w scalable deep learning/inference processor with

tetra-parallel mimd architecture for big data applications. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 80–81, 2015.

[144] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 92–104, 2015.

[145] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal Solid-State Circuits*, 52(1):127–138, 2016.

[146] Lei Yang, Weiwen Jiang, Weichen Liu, HM Edwin, Yiyu Shi, and Jingtong Hu. Co-exploring neural architecture and network-on-chip design for real-time artificial intelligence. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 85–90, 2020.

[147] Weiwen Jiang, Xinyi Zhang, Edwin H-M Sha, Lei Yang, Qingfeng Zhuge, Yiyu Shi, and Jingtong Hu. Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.

[148] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7370–7379, 2017.

[149] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction

method of multipliers. *Foundations and Trends in Machine learning*, 3(1):1–122, 2011.

[150] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, pages 1–14, 2015.

[151] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[152] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Conference on Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012.

[153] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[154] Guangcan Liu, Zhouchen Lin, Shuicheng Yan, Ju Sun, Yong Yu, and Yi Ma. Robust recovery of subspace structures by low-rank representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):171–184, 2013.

[155] Jian-Feng Cai, Emmanuel J Candès, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization (SIOPT)*, 20(4):1956–1982, 2010.

[156] Bing-Sheng He and Xiaoming Yuan. A class of ADMM-based algorithms for three-block separable convex programming. *Computational Optimization and Applications*, 2018.

[157] Caihua Chen, Bingsheng He, Yinyu Ye, and Xiaoming Yuan. The direct extension of ADMM for multi-block convex minimization problems is not necessarily convergent. *Mathematical Programming*, 155(1-2):57–79, 2016.

[158] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM International Multimedia Conference (MM)*, pages 675–678, 2014.

[159] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

[160] Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1984–1992, 2015.

[161] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *International Conference on Learning Representations (ICLR)*, 2016.

[162] Hongliang Yan, Yukang Ding, Peihua Li, Qilong Wang, Yong Xu, and Wangmeng Zuo. Mind the class weight bias: Weighted maximum mean discrepancy for unsupervised domain adaptation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2272–2281, 2017.

[163] Sinno Jialin Pan, James T Kwok, Qiang Yang, et al. Transfer learning via dimensionality reduction. In *AAAI Conference on Artificial Intelligence*, pages 677–682, 2008.

[164] Gilles Celeux and Gérard Govaert. A classification EM algorithm for clustering and two stochastic versions. *Computational statistics & Data analysis*, 14(3):315–332, 1992.

[165] Yves Grandvalet and Yoshua Bengio. Semi-supervised learning by entropy minimization. In *Conference on Neural Information Processing Systems (NIPS)*, pages 529–536, 2005.

[166] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2736–2744, 2017.

[167] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *International Conference on Learning Representations (ICLR)*, 2018.

[168] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *European Conference on Computer Vision (ECCV)*, pages 304–320, 2018.

[169] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2790–2799, 2019.

[170] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

[171] Yanghao Li, Naiyan Wang, Jianping Shi, Xiaodi Hou, and Jiaying Liu. Adaptive batch normalization for practical domain adaptation. *Pattern Recognition*, 80:109–117, 2018.

[172] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations (ICLR)*, 2017.

[173] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[174] Hang Zhang, Bei Yu, and Evangeline F. Y. Young. Enabling online learning in lithography hotspot detection with information-theoretic feature optimization.

In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 47:1–47:8, 2016.

[175] Haoyu Yang, Jing Su, Yi Zou, Yuzhe Ma, Bei Yu, and Evangeline F. Y. Young. Layout hotspot detection with feature tensor generation and deep biased learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 38(6):1175–1187, 2019.

[176] Haoyu Yang, Piyush Pathak, Frank Gennari, Ya-Chieh Lai, and Bei Yu. Detecting multi-layer layout hotspots with adaptive squish patterns. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 299–304, 2019.

# Publications Related to Thesis

[**1**] Yuzhe Ma, Wei Zhong, Shuxiang Hu, Jhih-Rong Gao, Jian Kuang, Jin Miao, Bei Yu, "A Unified Framework for Simultaneous Layout Decomposition and Mask Optimization", accepted by IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[**2**] Yuzhe Ma, Subhendu Roy, Jin Miao, Jiamin Chen, and Bei Yu, "Cross-layer Optimization for High Speed Adders: A Pareto Driven Machine Learning Approach", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. vol. 38, no. 12, pp. 2298–2311, 2019.

[**3**] Yuzhe Ma, Zhuolun He, Wei Li, Tinghuan Chen, Lu Zhang, Bei Yu, "Understanding Graphs in EDA: From Shallow to Deep Learning", ACM International Symposium on Physical Design, 2020. (Invited Paper)

[**4**] Yuzhe Ma, Ran Chen, Wei Li, Fanhua Shang, Wenjian Yu, Minsik Cho, Bei Yu, "A Unified Approximation Framework for Compressing and Accelerating Deep Neural Networks", IEEE International Conference on Tools with Artificial Intelligence, Portland, OR, Nov. 4–6, 2019. **(Best Student Paper Award)**

[**5**] Yuzhe Ma, Ziyang Yu, Bei Yu, "CAD Tool Design Space Exploration via Bayesian

Optimization", ACM/IEEE Workshop on Machine Learning for CAD, Alberta, Canada, Sep. 3–4, 2019.

[**6**] Yuzhe Ma, Haoxing Ren, Brucek Khailany, Harbinder Sikka, Karthikeyan Natarajan, and Bei Yu, "High Performance Graph Convolutional Networks with Applications in Testability Analysis", ACM/IEEE Design Automation Conference, Las Vegas, NV, June 2–6, 2019.

[**7**] Yuzhe Ma, Jhih-Rong Gao, Jian Kuang, Jin Miao, and Bei Yu, "A Unified Framework for Simultaneous Layout Decomposition and Mask Optimization", IEEE/ACM International Conference on Computer-Aided Design, Irvine, CA, Nov. 13–16, 2017.

[**8**] Yuzhe Ma, Xuan Zeng, and Bei Yu, "Methodologies for Layout Decomposition and Mask Optimization: A Systematic Review", IFIP/IEEE International Conference on Very Large Scale Integration, Abu Dhabi, UAE, Oct. 23–25, 2017. (Invited Paper)

[**9**] Subhendu Roy, Yuzhe Ma, Jin Miao, and Bei Yu, "A Learning Bridge from Architectural Synthesis to Physical Design for Exploring Power Efficient High-Performance Adders", IEEE/ACM International Symposium on Low Power Electronics and Design, Taipei, Taiwan, July 24–26, 2017.