Physical Design Automation for Next Generation ICs: Graph and Machine Learning based Approaches

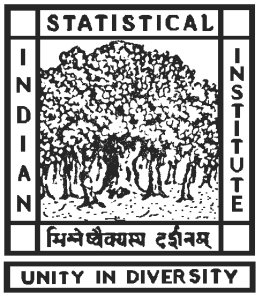
A thesis submitted for the degree of
Doctor of Philosophy

by

Kritanta Saha

Advisor

Prof. Susmita Sur-Kolay



Advanced Computing and Microelectronics Unit
Indian Statistical Institute
203 B. T. Road, Kolkata - 700108
West Bengal, India
August 2026

Dedicated to

To my family

Acknowledgements

The path leading to this thesis has been defined by challenges, growth, and meaningful accomplishments, and I remain deeply grateful to all who supported me throughout this journey.

I express my sincere appreciation to my supervisor, Prof. Susmita Sur-Koley, for her unwavering guidance throughout my Ph.D. Her insightful suggestions, constructive feedback, and steady encouragement have been instrumental in shaping the quality of this work. I am deeply grateful for her patience and belief in my abilities, especially during difficult phases of this journey.

I also extend my heartfelt thanks to Dr. Pritha Banerjee, who first introduced me to the field of VLSI Physical Design Automation. Her early encouragement during my M.Tech. studies inspired my interest in this area and continues to motivate me. I am equally thankful to the faculty members of Indian Statistical Institute for their teaching and guidance, which have greatly contributed to my academic development.

Every result presented in this thesis reflects the collective efforts of many. I sincerely thank Sudipta Paul and Mrinmoy Banik for their valuable contributions to our joint research papers and the stimulating discussions that enriched this work. I acknowledge using AI tools to assist with text formatting and to improve clarity.

I appreciate the support and camaraderie of my seniors and colleagues — Ritajit Majumdar, Debasmita Bhowmik, Manabendranath Mandal, and Santlal Prajapati — whose presence made this journey more enjoyable. I am also grateful to Sachi-da for his constant support, to Manas-da for his generous help, and to the office staff of the ACMU, ISI, for their continuous administrative and technical assistance.

I gratefully acknowledge the Indian Statistical Institute, Kolkata, for providing the fellowship that supported my Ph.D. research, and the TCS Foundation for awarding me the *TCS Research Scholar Program (RSP) Fellowship* (Cycle 15) from July 2019 to June 2023. I also thank the anonymous reviewers and experts whose insightful comments strengthened the quality of my research publications.

My heartfelt gratitude goes to my friend Debabrata Das and my cousin Rikan Saha for their unwavering emotional support and care. I also wish to acknowledge my

first computer teacher, Mr. Subrata Paul, whose early guidance fostered my lasting interest in this field.

Finally, I owe my deepest gratitude to my family. I thank my parents, Kamalesh Chandra Saha and Shanta Saha, my mother-in-law, Mita Malakar, and all my family members for their unconditional love, patience, and encouragement. My wife, Jayaita, has been my greatest strength. Her faith in me made this achievement possible. A special word of love goes to my daughter, Kriti Saha, whose tiny footsteps, laughter, and innocent eyes brought joy and hope during the most demanding moments. Without their love and support, this accomplishment would not have been possible.

To all mentioned here, and to anyone I may have inadvertently omitted, I offer my sincere gratitude.



Kritanta Saha

Abstract

Integrated Circuit (IC) technology has progressed significantly over the last six decades, culminating in billions of transistors on a chip. The widely used 193 nm immersion lithography process exceeds modern feature sizes, leading to layout pattern distortions and electrical violations. Mitigation techniques such as Multiple Patterning and Optical Proximity Correction become ineffective below 45 nm, prompting the development of several Next-Generation Lithography (NGL) methods such as Extreme Ultraviolet Lithography (EUVL), Multiple Electron Beam Lithography (MEBL).

MEBL employs thousands to millions of e-beams in parallel to overcome throughput limitations. A 2D layout is partitioned into vertical stripes by stitch-lines (SLs), each written by a distinct beam. Beam deflections near SLs cause distortions in the stitch-unfriendly region within a distance of ϵ of an SL. Minimizing these distortions during routing is critical. As routing is NP-hard in conventional lithography, the additional constraints of MEBL pose greater challenges, motivating this thesis to develop enhanced routing strategies for MEBL in VLSI physical design automation.

This thesis first addresses the reduction of stitch-induced distortions during global routing. A stitch-avoiding global routing framework that integrates stitch-lines into the routing process is designed. It combines net-shifting strategies with stitch -constrained Integer Linear Programming (ILP) and maze routing to achieve a significant reduction in stitch-line violations compared to earlier approaches.

For further reductions, a dedicated full-routing framework by introducing a stitch-avoiding detailed router is designed, which classifies nets by characteristics and applies a multi-stage routing strategy. Layer assignment is integrated within detailed routing, while segment-to-track assignment is formulated as a maximum weighted independent set problem on a segment conflict graph, with weights derived from stitch-line proximity. Combining the stitch-avoiding global and detailed routers yields substantial improvement over conventional routing flows.

Beyond lithography-specific routing, the backbone of global routing is a Rectilinear Steiner Minimum Tree (RSMT), which connects a set of n pins of a net on a

2D plane, augmented with Steiner points, with minimum total wirelength under the Manhattan metric. An optimal RSMT construction is NP-hard, existing methods either achieve optimality with exponential/sub-exponential complexity, or produce approximate solutions by using heuristics or reinforcement learning. Here, a machine learning approach for finding an RSMT of a net is proposed, followed by another ML method for the construction of RSMTs of multiple nets concurrently.

The input for RSMT construction is the Hanan grid of the pin locations, which can be represented as an image. Supervised training of a Conditional Generative Adversarial Network (cGAN) with data from an existing exponential-time optimal RSMT method enables the generator to produce valid RSMTs with minimal wirelength. To the best of our knowledge, this is the first application of image-to-image translation for RSMT construction, achieving notable improvements in wirelength and runtime over heuristic and RL-based methods.

For concurrent RSMT construction, individual Hanan grids of multiple nets are merged into a combined grid graph with node and edge labels for each net. An ILP-based formulation is used to generate group-wise edge-overlap-minimized combined RSMT that acts as the ground truth for our proposed hybrid Graph Neural Network (GNN) model. Experiments show a significant reduction in runtime, group-wise edge-overlap, and wirelength compared to multiple baseline methods. To the best of our knowledge, this is the first successful attempt for the concurrent construction of RSMTs of multiple nets using a hybrid GNN model.

Thus, this thesis advances routing for MEBL and efficient RSMT construction for global routing to improve the performance of modern ICs.

Contents

1 Introduction and the Scope of the Thesis **1**

1.1 Introduction 2

1.2 Physical Design Flow 9

1.3 Next-Generation Lithography for IC Fabrication 11

1.3.1 Extreme Ultraviolet Lithography 11

1.3.2 Nanowire 12

1.3.3 Electron Beam Lithography 12

1.3.4 Multiple Electron Beam Lithography 12

1.4 Motivation and Scope of this thesis 13

1.4.1 Routing Challenges in Multiple E-Beam Lithography 14

1.4.2 Challenges in RSMT Construction and Machine Learning-Based Solutions 16

1.5 Organization of the Thesis 17

2 Background and Related Works **19**

2.1 Introduction 20

2.2 Physical Design Techniques for Photolithography 21

2.2.1 Partitioning 21

2.2.2 Floorplanning 23

2.2.3 Placement 23

2.2.4 Routing 24

2.2.5 Extraction and Physical Verification 29
2.3 Photolithography: Enhancements and Constraints 30
2.3.1 Optical Proximity Correction 30
2.3.2 Multiple Patterning 30
2.3.3 Limitations of OPC and MP 33
2.4 Next Generation Lithography Techniques and Limitations 34
2.4.1 Extreme Ultraviolet Lithography 34
2.4.2 Nanowire 36
2.4.3 Electron Beam Lithography 37
2.4.4 Multiple Electron Beam Lithography 38
2.5 Existing Approaches to Mitigate Stitch-Line Violations 40
2.5.1 Comparative Analysis of Existing MEBL-Aware Methodologies 44
2.5.2 Research Gap and Motivation 46
2.6 Baseline Routing Framework 46
2.6.1 BoxRouter: A Traditional Global Router 47
2.6.2 RegularRoute: A Traditional Detailed Router 49
2.7 Machine Learning for Physical Design Automation 51
2.7.1 ML for Floorplanning and Placement Heuristics 52
2.7.2 ML for Global Placement and Routability Optimization 52
2.7.3 ML for Routing, Routing Congestion and Routability Prediction 53
2.7.4 ML for Crosstalk and Signal Integrity Prediction 55
2.7.5 ML Methods in Lithography Hotspot Detection 55
2.7.6 Categories of Machine Learning Methods for Physical Design Problems 57
2.8 Existing Approaches for RSMT Construction 59
2.8.1 Comparative Analysis of Existing RSMT Construction Methodologies 62
2.8.2 Research Gap and Motivation 62

2.9 Summary 62

3 Stitch-avoiding Global Routing for Multiple E-Beam Lithography 64

3.1 Introduction 65
3.2 Motivation 66
3.3 Preliminaries and Problem Formulation 66
3.3.1 Preliminaries 66
3.3.2 Problem Formulation 70
3.4 Overview of Our Proposed Global Router for MEBL 71
3.5 Phase-I: Stitch-avoiding Steiner Minimum Tree (SSMT) 71
3.6 Phase-II: Multistage Stitch-avoiding Global Routing 75
3.6.1 Stitch-avoiding Global Routing of Two-Pin Flat Nets 76
3.6.2 Stitch-avoiding Global Routing for Multi-Pin Nets and TPBN 76
3.6.3 Stitch-constrained Maze Routing 89
3.7 Evaluation of our Stitch-avoiding Global Router 94
3.7.1 Discussion on the Optimality and Scalability of the Proposed Routing Framework 98
3.7.2 Limitations and Correctness of the Proposed Routing Framework 100
3.8 Concluding Remarks 101

4 Stitch-avoiding Detailed Routing for Multiple E-Beam Lithography 102

4.1 Introduction 103
4.2 Motivation 103
4.3 Preliminaries and Problem Formulation 104
4.3.1 Preliminaries 104
4.3.2 Problem Formulation 107
4.3.3 Net Classification 108
4.4 Overview of Our Proposed Detailed Routing Framework for MEBL . 110

4.5 Detailed routing for Single-Cell Confined Nets (SCNs) with stitch avoidance . . . 113
4.6 Stitch-avoiding Detailed Routing of MCNs . . . 114
4.6.1 Routing Vertical Segments of MCNs . . . 115
4.6.2 Routing of horizontal segments in MCMPNs . . . 126
4.6.3 Routing of MCSPNs having all HSegments in one HPanel . . . 128
4.6.4 Discussion on the Computational Overhead of the Proposed Framework . . . 130
4.7 Experimental Results . . . 131
4.8 Conclusion . . . 135

5 Constructing Rectilinear Steiner Minimum Tree with Conditional Generative Adversarial Network 136

5.1 Introduction . . . 137
5.2 Motivation . . . 139
5.3 Preliminaries and Problem Formulation . . . 140
5.3.1 Preliminaries . . . 140
5.3.2 Problem Formulation . . . 141
5.4 Proposed RSMT Construction Methodology . . . 141
5.4.1 Overview . . . 141
5.4.2 Image Encoding Mechanisms . . . 143
5.4.3 cGAN Model Architecture for RSMT Construction . . . 153
5.4.4 Loss Functions . . . 155
5.5 Experimental Setup . . . 156
5.5.1 Generation of Training Dataset . . . 158
5.5.2 Generation of Test Dataset . . . 159
5.5.3 Selection of Baseline Methods . . . 159
5.6 Experimental Results . . . 160
5.6.1 Computational Cost of the Proposed Framework . . . 164

5.6.2 Generalization Capability of the Proposed Framework 165
5.6.3 Scalability of the Proposed Framework 166
5.6.4 Prediction Reliability 166
5.6.5 Model Interpretability 167
5.7 Concluding Remarks 168

6 Concurrent Generation of RSMTs for Multiple Nets using Graph Neural Network 169

6.1 Introduction 171
6.2 Motivation and Major Contributions 172
6.3 Preliminaries and Problem Formulation 174
6.3.1 Preliminaries 175
6.3.2 Problem Formulation 179
6.4 Overview of Proposed Methodology 180
6.5 Generation of Dataset 182
6.5.1 Net Grouping with Fixed Group Size 182
6.5.2 Combined Hanan Grid Graph Construction 185
6.5.3 Edge-Overlap-Minimized Combined RSMT Construction . . . 187
6.6 Our Hybrid GNN Model Architecture and Loss 191
6.6.1 Input Encoding using Linear Encoders 192
6.6.2 Message Passing Layers 195
6.6.3 Jumping Knowledge Aggregation 198
6.6.4 Global Attention, Virtual Slots, and FiLM Modulation 199
6.6.5 Adaptive Edge-Level Attention 200
6.6.6 Modeling Inter-Net Dependencies 201
6.6.7 Node Classifiers 202
6.6.8 Edge Classification Head 203
6.6.9 Discussion on the Selection of the GNN Architecture 205
6.6.10 Loss Generation 206

6.7 Formation of Individual Predicted RSMTs 207
6.7.1 Branch-Aware Edge Probability Smoothing 208
6.7.2 Priority-Guided Tree Construction 208
6.8 Experimental Setup 208
6.8.1 Selection of Hyperparameters 209
6.8.2 Details of Training Dataset 209
6.8.3 Details of Test Dataset 211
6.9 Experimental Results 211
6.9.1 Generalization Capability of the Proposed Framework 216
6.9.2 Interpretability of Learned Representations 217
6.10 Concluding Remarks 218

7 Conclusions and Future Directions **219**
7.1 Summary 220
7.2 Future Directions 222

Bibliography **225**

LIST OF PUBLICATIONS RELATED TO THIS THESIS

The thesis is based on the following publications of the author.

In Journals

[1] **Kritanta Saha**, Mrinmoy Banik, Pritha Banerjee and Susmita Sur-Kolay, "Concurrent Generation of RSMTs for Multiple Nets using Graph Neural Network," submitted to Integration, the VLSI Journal.

In Peer-reviewed International Conference Proceedings

[2] **Kritanta Saha**, Pritha Banerjee and Susmita Sur-Kolay, "Constructing Rectilinear Steiner Minimum Tree with Conditional Generative Adversarial Network," 2025 38th International Conference on VLSI Design and 2025 24th International Conference on Embedded Systems (VLSID), Bangalore, India, 2025, pp. 351-356. (*Best Paper Award*) doi: 10.1109/VLSID64188.2025.00073

[3] **Kritanta Saha**, Pritha Banerjee and Susmita Sur-Kolay, "Stitch-avoiding Detailed Routing for Multiple E-Beam Lithography," 2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC), Patras, Greece, 2022, pp. 1-6, doi: 10.1109/VLSI-SoC54400.2022.9939588

[4] **Kritanta Saha**, Sudipta Paul, Pritha Banerjee and Susmita Sur-Kolay, "Stitch-avoiding Global Routing for Multiple E-Beam Lithography," 2022 35th International Conference on VLSI Design and 2022 21st International Conference on Embedded Systems (VLSID), 2022, pp. 138-143, doi: 10.1109/VLSID2022.2022.00037

List of Figures

1.1 Overview of VLSI Design Flow (Courtesy [She99]) . . . 3
1.2 Photolithography (Courtesy [Ham25]) . . . 7
1.3 Sub-wavelength lithographic gap (Courtesy [CC09b]) . . . 8
1.4 Physical Design Flow (Courtesy [She99]) . . . 10
1.5 Example RSMT of a 10-degree net . . . 14

2.1 Partitioning of a Circuit (Courtesy [She99]) . . . 22
2.2 Block placement (Courtesy [She99]) . . . 24
2.3 Grid Graph Model (Courtesy [She99]) . . . 26
2.4 Major pattern distortions and improvements with OPC . . . 31
2.5 Multiple Patterning . . . 32
2.6 The photo-reflective layer on the mask redirects light, and the pattern is generated using sections with light-absorbing materials. (Courtesy [FC12]) . . . 35
2.7 Nanowire Lithography . . . 36
2.8 Electron Beam Lithography and throughput enhancement . . . 38
2.9 (a) Stitch-lines in a layout, (b) patterns cut by stitch-lines, (c) pattern distortion due to overlay error, and (d) violations due to a stitch-line. [SPBSK22] . . . 39
2.10 Box Routing concept, (a) Working principle of BoxRouting, (b) strategies of Routing. (Courtesy [CP06]) . . . 47
2.11 BoxRouting flow. (a) Initial hotspot box. (b) Wires in box i. (c) PILP routing. (d) AMR for unrouted wires. (e–f) Box expansion and rerouting. (Courtesy [CP06]) . . . 48
2.12 General PILP formulation. (Courtesy [CP06]) . . . 49

2.13 Track, segment, and panel definitions (Courtesy [ZC12]) 50

3.1 Overview of our proposed Global Routing Method for MEBL 72

3.2 (a) Steiner branch between $SP_{i,1}$ and $SP_{i,2}$ causing via and vertical routing violations, (b) violations removed after SBS, (c) Steiner point $SP_{i,1}$ causing short polygon violation, and (d) short polygon removed after SPS. 73

3.3 (a) A Steiner branch connecting $SP_{i,1}$ and $SP_{i,2}$ leads to via-related and vertical routing conflicts, (b) these conflicts are eliminated after applying SBS, (c) Steiner point $SP_{j,1}$ introduces a short-polygon violation, and (d) this violation is resolved through SPS with an added wirelength of one unit. 74

3.4 Illustration of two patterns corresponding to a net: (a) *Category* 1, (b) *Category* 2 78

3.5 (a) Computation of $\alpha_{i,1}$ and $\alpha_{i,2}$ based on endpoint position relative to stitch-line; (b) Making path $X_{i,2}$ violation-free using an additional Steiner point 80

3.6 Generalized stitch-constrained ILP to route two-pin bend nets and two-pin segments present inside the k^{th} box 85

3.7 Example of our Stitch-constrained ILP with three nets inside the kth box 87

3.8 Twelve possible patterns considered during Retrack 90

4.1 (a) Illustration of the G-cell along with the horizontal and vertical panels; (b) depiction of the Stitch Critical Region. 105

4.2 Various net types are illustrated: Net A (blue) corresponds to a Single-Cell Confined Net (SCN), Net B (green) corresponds to a Multi-Cell Single-Panel Net (MCSPN), and Net C (purple) corresponds to a Multi-Cell Multi-Panel Net (MCMPN). 109

4.3 Overview of the proposed stitch-avoiding detailed routing framework 111

4.4 (a) A single-trunk V-tree is formed by placing a trunk at the median x-location of the pins, which results in several short polygons. (b) Shifting the vertical trunk locally removes some of these short polygons, although the violation near pin $P_5^{N_i}$ remains. (c) Adding a new HSegment along with a via eliminates the short polygon at $P_5^{N_i}$. (d) The via is then connected to the vertical trunk through a horizontal segment routed on an upper HLayer. 115

4.5 Formation of the Segment Conflict Graph (SCG); (a) VSegments located in panel Π_k; (b) Current occupancy of Π_k; (c) The resulting SCG. Black edges denote conflicts among alternative choices of a single segment, while magenta edges indicate conflicts between options belonging to distinct segments. 118

4.6 Segment splitting and routing of delayed Segments under various scenarios: – Case I: (a) Placing Ψ^{N_1} on T_3 leads to a short–polygon issue at $End1$; (b) introducing a new via resolves the problem. – Case II: (a) The selected VTracks cause a short polygon near $End1$; (b) the segment is divided; (c) the resulting pieces, routed using additional vias, remove the violation. – Case III: (a) The assigned VTracks generate a short–polygon violation at $End1$; (b) the segment is split; (c) routing the split parts with a new via clears the error. – Case IV: (a) The chosen VTracks produce short polygons at both $End1$ and $End2$; (b) the segment is broken into smaller portions; (c) routing these portions using a new via resolves both violations. 124

4.7 Assignment of Partial Vertical Segment: (a) The vertical segment Ψ^{N_i} is partially assigned on T_1;; (b) Candidate via sites for linking to a Horizontal Segment; (c) Feasible via locations at both endpoints of Ψ^{N_1} when mapped to either T_1 or T_2. 125

5.1 (a) Hanan grid with pin positions of a net (b) Optimal RSMT containing only Hanan grid edges and Steiner points 138

5.2 Overview of our `pix2pix` based cGAN model with input and ground truth RSMT images 142

5.3 Topology-preserving mapping of pin location and length of Hanan edge to a pixel intensity for Input image generation: (a) Hanan grid with pins, (b) corresponding matrix M of size 32×32 (c) corresponding input image encoding (pixels with small intensities may appear to be black however, contain valid information) (d) Ground Truth RSMT (e) Modified Matrix M to represent the Ground Truth RSMT (f) corresponding ground truth RSMT image encoding (pixels with small intensities may appear to be black however, contain valid information). 146

5.4 (a) Matrix M of size 32×32 (b) corresponding input image encoding (pixels with small intensities may appear to be black, however, contain valid information) 151

5.5 (a) Modified Matrix M to represent the Ground Truth RSMT (b) corresponding ground truth RSMT image encoding (pixels with small intensities may appear to be black, however, contain valid information). 152

5.6 Architecture of the U-Net-based generator model 154

5.7 Architecture of the PatchGAN-based discriminator model 155

5.8 Training samples for category I – the upper row shows input images and the corresponding ground truth RSMT images are in the bottom row. (Pixels with small intensities may appear to be black, however, contain valid information.) 159

5.9 Generator loss encountered during the training process 161

5.10 Discriminator loss encountered during the training process, where the green line denotes the discriminator loss for classifying real RSMT images and the blue line denotes the discriminator loss for classifying generated RSMT images 162

5.11 Progress of the generator model after 5 training epochs. The upper row contains the input image, the middle row shows the RSMT obtained after 5 epochs, and the lower row presents the actual RSMTs. Pixels with minimal intensity can look black while still encoding valid information. 162

5.12 Results of Testing. The top row shows input images, the middle row shows generated RSMTs, and the bottom row shows the ground truth RSMTs. Pixels with small intensities may appear to be black. However, they contain valid information. 163

6.1 (a) Hanan grid with pin positions of a net (b) Optimal RSMT containing only Hanan grid edges and Steiner points (c) Combined Hanan grid with pins of two nets (d) Corresponding combined ground truth optimal RSMTs of two nets 172
6.2 Outline of the proposed method: (a) Generation of training and validation datasets with training of our hybrid GNN model (b) Test dataset generation and RSMT prediction. Pink blocks represent our novel approaches for concurrent RSMT construction of multiple nets, yellow blocks indicate our application-specific contributions in model design and training, and cyan blocks denote general methods. 181
6.3 Combined Hanan grid graph with node and edge feature vectors . . . 186
6.4 Generalized ILP for edge overlap minimization of combined RSMTs in each group 188
6.5 Combined RSMT with node and edge labels 190
6.6 Our Hybrid GNN Model Architecture by considering the number of message passing layers $L = 10$, multi-hop interval $\mathcal{I} = 4$, and following Equation 6.17 and Equation 6.18. 193

List of Tables

1.1 Overview of the VLSI Design Flow [She99] 5

2.1 Comparison of Existing RSMT Generation Methods 61

3.1 List of Symbols and Their Meanings 68
3.2 Net and Segment Data for Weight Calculation 79
3.3 Example of assigning weights $\alpha_{i,1}$ and $\alpha_{i,2}$ using Algorithm 3.1, based on the position of pins and stitch-lines. The table covers all three cases handled by the algorithm. 83
3.4 Faraday benchmark circuits of ICCAD04 [oCADI21] 94
3.5 Number of Steiner Branches (SB) and Steiner Points (SP) before and after performing SBS and SPS 95
3.6 Comparison of Our Proposed Stitch-avoiding Global Router with Traditional Global Router 96
3.7 Effectiveness of the proposed Stitch-avoiding Global Router after detailed routing 97

4.1 List of Symbols and Their Meanings 105
4.2 Faraday benchmark circuits of ICCAD04 [oCADI21] 132
4.3 Routing parameter values 132
4.4 Experimental result showing the effectiveness of our Proposed Stitch-avoiding Detailed Router 134

5.1 List of Symbols and Their Meanings 140
5.2 Coordinates of the pins of a 10-degree net 143
5.3 Computation of transformed x-coordinates 148

5.4 Training dataset . . . 158
5.5 Validation dataset . . . 158
5.6 Test dataset . . . 160
5.7 Comparison of average percentage error in wirelength of our method with GeoSteiner [JWWZ18], FLUTE [CW07], and REST [LCY21] . . . 164
5.8 Comparison of RSMT Generation Time (sec) of our method with GeoSteiner [JWWZ18], FLUTE [CW07], REST [LCY21] . . . 164

6.1 List of Symbols and Their Meanings . . . 176
6.2 Pin coordinates corresponding to a 10-degree net N_1 . . . 185
6.3 Pin coordinates corresponding to a 10-degree net N_2 . . . 186
6.4 Pin coordinates corresponding to a 6-degree net N_3 . . . 186
6.5 Decision variables used in the ILP formulation. . . . 188
6.6 Test Dataset . . . 211
6.7 Comparison of avg. % error in wirelength of our method with Baseline 1: GeoSteiner[JWWZ18] + IEOM*, Baseline 2: Flute[CW07] + IEOM*, Baseline 3: REST[LCY21] + IEOM* and Baseline 4: cGAN-RSMT [SBSK25] + IEOM* . . . 213
6.8 Comparison of RSMT Generation Time (sec) of our method with Baseline 1: GeoSteiner [JWWZ18] + IEOM*, Baseline 2: Flute [CW07] + IEOM*, Baseline 3: REST [LCY21] + IEOM* and Baseline 4: cGAN-RSMT [SBSK25] + IEOM* . . . 214
6.9 Comparison of group-wise avg. % overlap mitigation of our method with Baseline 1: GeoSteiner [JWWZ18] + IEOM*, Baseline 2: Flute [CW07] + IEOM*, Baseline 3: REST [LCY21] + IEOM* and Baseline 4: cGAN-RSMT [SBSK25] + IEOM* . . . 215

CHAPTER

ONE

INTRODUCTION AND THE SCOPE OF THE THESIS

Contents

1.1 Introduction **2**

1.2 Physical Design Flow **9**

1.3 Next-Generation Lithography for IC Fabrication **11**

1.3.1 Extreme Ultraviolet Lithography 11

1.3.2 Nanowire 12

1.3.3 Electron Beam Lithography 12

1.3.4 Multiple Electron Beam Lithography 12

1.4 Motivation and Scope of this thesis **13**

1.4.1 Routing Challenges in Multiple E-Beam Lithography 14

1.4.2 Challenges in RSMT Construction and Machine Learning-Based Solutions 16

1.5 Organization of the Thesis **17**

1.1 Introduction

Integrated Circuit (IC) technology has advanced tremendously over time. It began with Small-Scale Integration (SSI), where only a few transistors could be placed on a chip. Over successive generations, technology progressed, and today we have Very or Ultra Large-Scale Integration (VLSI / ULSI). As a result, a modern chip can contain billions of transistors [Wol08]. An IC is built from multiple electronic components arranged in precise material layers on a silicon *wafer* [She99]. The IC design process begins with a circuit description, which is transformed into a geometric representation known as a *layout*. This layout comprises planar geometric shapes across multiple layers and undergoes rigorous checks to ensure compliance with electrical, physical, and manufacturing constraints [Wol08]. The verified layout is translated into design files, which are then processed by an optical pattern generator to create masks. These masks serve as templates in successive *photolithography* steps [Mor12, Che25], where circuit patterns are transferred onto the wafer surface for fabrication.

The VLSI design flow refers to the systematic methodology employed to transform high-level system requirements into a fully functional IC. It provides a structured framework for managing the increasing complexity of modern chips, ensuring that the final design satisfies constraints on performance, area, power, and cost [WH15]. Designing and fabricating highly complex circuits becomes impractical without a well-established design flow [She99]. The different stages of the VLSI design flow are shown in Figure 1.1, and a brief description is given below:

1. **System Specification** – This stage defines the functional requirements, power, size, performance, and technology constraints of the system. It represents a compromise between technological capabilities, market demands, and economic feasibility [Sau23].
2. **Architectural Design** – At this stage, the overall system architecture is established, including decisions on processor type, cache structures, and pipeline organization. The outcome is a Micro-Architectural Specification (MAS), which provides early estimates of power, performance, and die area.
3. **Behavioral (Functional) Design** – The functional units of the system and their interactions at a high level of abstraction are specified in this phase. It

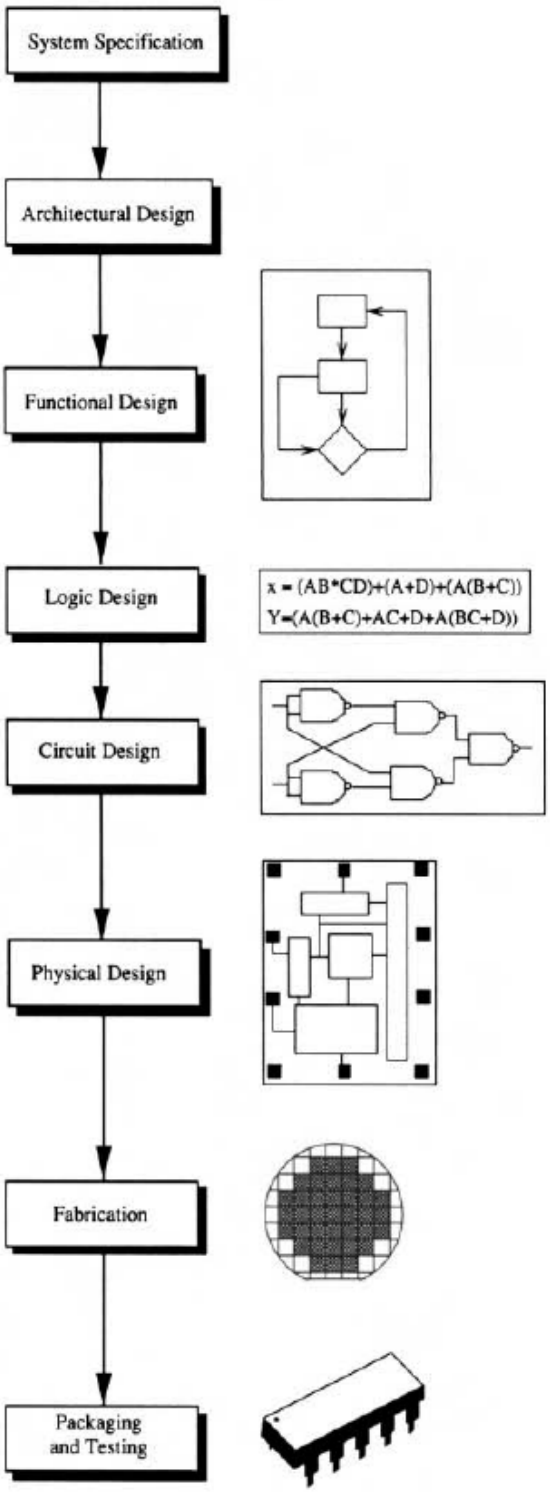


Figure 1.1: Overview of VLSI Design Flow (Courtesy [She99])

focuses on input-output behavior and timing without prescribing internal implementation details, thereby facilitating early validation and debugging.

4. **Logic Design** – Hardware Description Languages (HDLs) such as Verilog or VHDL are used to translate the behavioral description into a Register Transfer Level (RTL) representation [LMMS17]. This representation is optimized and verified to ensure functional correctness [PLL+24].
5. **Circuit Design** – Transistor-level circuits in the form of netlists are obtained by mapping the RTL description. Considerations of timing, speed, and power are addressed through detailed circuit simulation at this stage [PLL+24].
6. **Physical Design** – From the circuit representation, a geometric layout specifying the placement of cells, gates, and interconnections on silicon is derived. A *net* refers to a set of interconnected circuit elements that share the same electrical connection. An IC consists of several nets that form a *netlist*. The physical design process includes placement, routing of the netlist, and design-rule verification, and it is often the most computationally intensive phase of the flow [KLMH11, QXZ+23].
7. **Fabrication** – The verified layout is converted into photolithographic masks, which are subsequently employed to manufacture the integrated circuit on silicon wafers using deposition, diffusion, and etching processes [WH15].
8. **Packaging, Testing, and Debugging** – Fabricated semiconductor substrate is cut into discrete chips, each of which is packaged in a suitable form and rigorously tested to confirm compliance with the design specifications [Sau23].

Table 1.1 gives a summary. Although every stage is important, *physical design phase* is especially significant because it is responsible for the transformation of a circuit specification into a manufacturable layout [MHK12]. It connects the logical description of a circuit to its actual silicon implementation. The choices made at this stage have a direct impact on chip performance, power usage, area, and ease of manufacturing, which makes this step very important.

Physical design is the process to convert a netlist of a circuit into a geometric layout, where components and their interconnections are represented as shapes across multiple layers, subject to fabrication design rules. The extremely small dimensions of modern components make physical design a highly complex and error-prone process.

Table 1.1: Overview of the VLSI Design Flow [She99]

Step	**Objective**	**Output**
System Specification	Define functionality, size, power, and performance requirements	System-level specifications
Architectural Design	Establish processor type, pipelines, cache, etc.	Micro-Architectural Specification (MAS)
Behavioral Design	Specify functional units and interactions at high abstraction	Functional specification, timing diagrams
Logic Design	Translate behavior into RTL using HDLs, optimize logic	RTL description verified for correctness
Circuit Design	Map RTL to transistor-level circuits with timing/power validation	Netlist with transistor-level details
Physical Design	Convert netlist into geometric layout with placement and routing	Verified chip layout ready for fabrication
Fabrication	Manufacture IC using masks and semiconductor processes	Silicon wafers containing ICs
Packaging, Testing, Debugging	Dice, package, and test chips for compliance	Packaged and tested integrated circuits

For example, today's microprocessors can contain more than *50 billion transistors* built at technology nodes as small as *3 nanometers*. The physical design process is divided into multiple phases due to complexity with verification at each stage [She99, Cha24].

Data structures and algorithms that support the process of physical design are researched, developed and deployed in *VLSI Physical Design Automation*. Its primary goal is to determine the optimal placement of the device, and efficient interconnection methods that achieve the required functionality and performance. Given the high cost and limited availability of wafer area, algorithms are required to optimize space utilization to improve yield and reduce manufacturing expenses. Moreover, the placement of devices and routing directly influences the overall performance of the chip. [She99, Cha24].

Automated layout synthesis tools enable faster design with reduced effort but typically incur area and performance penalties, making them suitable for lower-performance or time-critical designs. Manual layout, though slower and labor-intensive, generally provides better performance and area efficiency, but its effectiveness decreases as design size and complexity grow. In almost every stage of physical design, reliance is placed on Computer-Aided Design (CAD) tools, several of which are already partially or fully automated. This level of integration has been achieved through advancements in *Electronic Design Automation (EDA)*, *photolithography* [Che25], and manufacturing methods, along with the growing application of *machine learning techniques* [YPMZ15, Kah22] that aid in layout optimization, improve efficiency, and improve overall design quality.

Physical design algorithms are required to ensure compliance of the generated layout with the fabrication rules that delineate the tolerances of the manufacturing process. Furthermore, scalability and the ability to manage large-scale designs are essential requirements for these algorithms. The use of efficient algorithms facilitates faster design turnaround and iterative layout optimization, leading to improved performance, reduced cost, and enhanced manufacturability [She99].

The overview of various phases of the physical design flow is described in Section 1.2. Each of these phases typically involves NP-hard problems, which have traditionally been addressed through heuristic and graph-theoretic approaches. The advancement in machine learning techniques and computational power enabled the exploration of data-driven methods for tackling these challenges [Kah22]. However, the problems encountered in physical design are not limited to standard tasks such as classification, regression, or clustering; rather, they are complex optimization problems

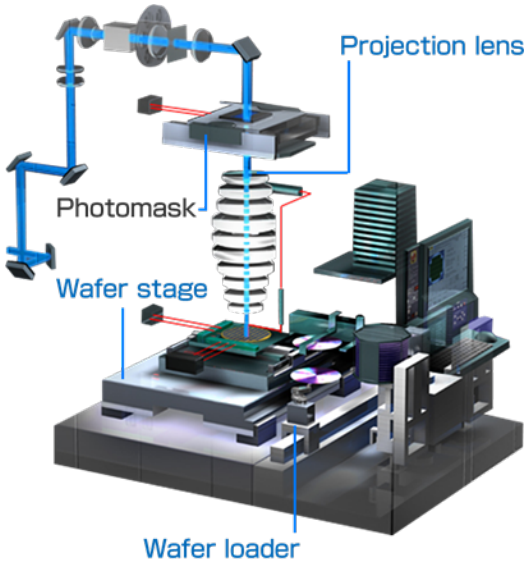


Figure 1.2: Photolithography (Courtesy [Ham25])

closely tied to design-specific constraints. Consequently, applying machine learning to these problems is non-trivial and requires specialized formulations. The existing machine learning methods in the literature are reviewed in Section 2.7.

The fabrication phase is another highly important stage in the VLSI design flow. *Photolithography* [Mac08] is an IC fabrication technology in which light is projected through masks to transfer geometric patterns onto the wafer surface, as illustrated in Figure 1.2. A light-sensitive material called photoresist is used to coat the wafer. Then an ultraviolet (UV) light is projected through a mask containing the desired pattern. The exposed regions of the photoresist undergo chemical changes, allowing selective etching or deposition. Accurate control of feature dimensions and inter-feature spacing is critical, since minor variations may compromise device functionality and manufacturing yield [KG21, KGTC23].

Immersion lithography is an advanced form of photolithography that introduces a high–refractive-index liquid between the lens and the wafer. The introduction of the liquid elevates the optical system's effective numerical aperture (NA), consequently shortening the effective wavelength of light and enhancing resolution. This approach enables the patterning of smaller feature sizes using 193 nm light without requiring an

immediate transition to a new light source. However, as IC fabrication technologies continue to advance, physical design has become increasingly complex, and layout features have scaled deep into the nanometer regime. As a result, 193 nm *immersion lithography*, widely adopted in IC fabrication, has reached its fundamental limits, since its wavelength is mismatched with the extremely small feature sizes of modern devices [JLY+22].

Sub-wavelength lithographic gap [YYDP15] refers to the difference between the printing wavelength and the size of the layout feature, as shown in Figure 1.3, which causes pattern distortions and degrades the quality of IC designs. To overcome this problem, mitigation techniques like *Multiple Patterning* (MP) [YYDP15] and *Optical Proximity Correction* (OPC) [HLC10] have been proposed. A concise overview of these existing methods is provided in Section 2.3. MP enables patterning of features well below the 193 nm wavelength limit—down to the $20 - 30$ nm range—by decomposing layouts into multiple exposures. OPC improves fidelity by compensating for optical distortions, allowing accurate printing of features close to 1/4–1/5 of the wavelength. Together, these methods extend immersion lithography well beyond its conventional resolution limits, enabling the fabrication of increasingly advanced technology nodes. However, they are also unable to cope with the modern nanometer IC design. As a result, several *Next-Generation Lithography* techniques [HL18] have been developed.

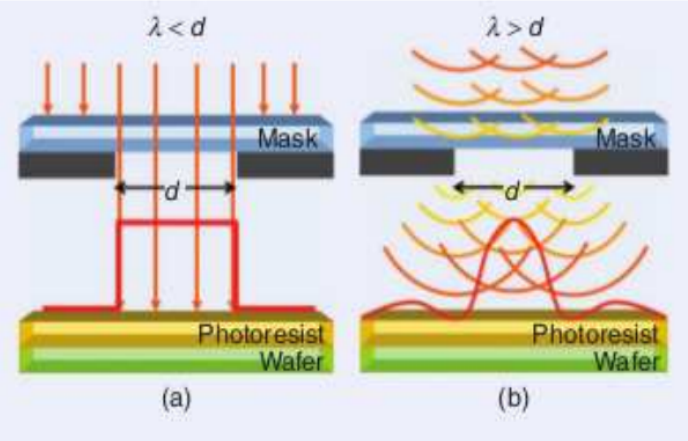


Figure 1.3: Sub-wavelength lithographic gap (Courtesy [CC09b])

Next-Generation lithography refers to emerging semiconductor patterning technologies designed to overcome the resolution and scaling limitations of conventional photolithography, particularly 193 nm immersion lithography [FCC12a]. Over the years, several *Next-Generation lithography* methods have been introduced, as follows:

- *Extreme Ultraviolet Lithography* (EUVL) [FC12, PBSK21]
- *Nanoimprint lithography* [RSV11, HWL+23]
- *Electron Beam Lithography* (EBL) [PBM00, YYGP13]
- *Multiple Electron Beam Lithography* (MEBL) [LFC15, HF20], etc.

A concise overview of Next-Generation Lithography (NGL) techniques is provided in Section 1.3. Compared to conventional photolithography, these methods enable writing of much smaller-scale features on silicon wafers, resulting in chips that are faster, more powerful, and more compact. They also improve pattern accuracy, which reduces defects and improves the overall reliability of the device. Furthermore, NGL techniques allow for higher transistor and interconnect density, which is crucial for increasing functionality within the same chip area [HL18]. Despite these benefits, each technique introduces significant technical and manufacturing challenges that must be addressed before it can be implemented in large-scale production.

1.2 Physical Design Flow

In the physical design flow, a circuit diagram is used as input to produce the corresponding circuit layout as output. This transformation is accomplished across a series of stages encompassing partitioning, floorplanning, placement, routing, and compaction [She99]. Figure 1.4 illustrates the overall flow of these stages. Boolean circuit design acts as the input to the physical design phase. The following gives a very brief description of each stage of physical design. The detailed description is given in Section 2.2.

- **Partitioning:** Divides large circuits into smaller modules called blocks at system, board, and chip levels. It is formulated as a hypergraph partitioning problem with objectives to minimize interconnections, delay, and cut-size while satisfying terminal, area, and partition constraints [She99, KLMH11].

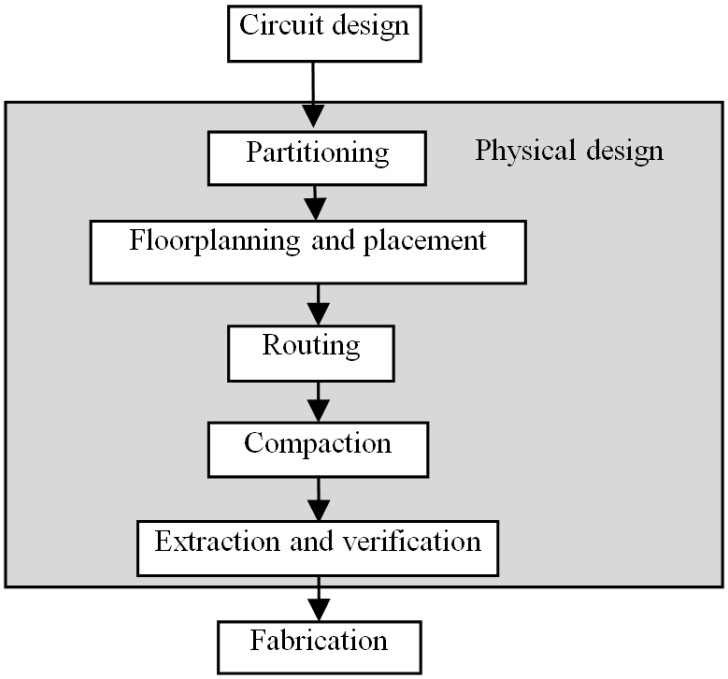


Figure 1.4: Physical Design Flow (Courtesy [She99])

- **Floorplanning:** Determines feasible block dimensions and aspect ratios. It ensures blocks are shaped and arranged for routability while minimizing overall layout area [VNA+07].
- **Placement:** Assigns blocks precise positions on the chip to minimize area and wirelength. Performance-driven placement reduces delay by minimizing critical paths, often using heuristic or ML-based methods [VPC07, MGY+21].
- **Routing:** Completes netlist interconnections through global routing, layer assignment, track assignment, and detailed routing [LFC15]. Minimizes wirelength, satisfies design rules, and resolves congestion using heuristics [PXZC12, CSD+24].
  - **Global Routing:** A standard global routing method uses graph models (commonly grid graphs) to plan approximate paths. Several state-of-the-art global routers use Rectilinear Steiner Minimum trees (RSMTs) that connect the pins of nets while minimizing wirelength [CP06, LCY21].

  - **Layer Assignment:** In modern ICs, multiple metal layers are required to accommodate the large number of routing segments needed for detailed routing connections. This method maps net segments onto available metal layers. It optimizes routability by minimizing congestion, wirelength, and via usage while meeting capacity constraints [DAL15].
  - **Track Assignment:** It allocates routed segments to predefined horizontal/vertical tracks present in each metal layer. It also ensures spacing, reduces cross-talk, and improves timing reliability before detailed routing [CP06, LCZ22].
  - **Detailed Routing:** This method assigns exact wire geometries within allowed regions. It connects all terminals while minimizing wirelength and vias, adhering to spacing, layer, and design rules. [KSKM18, ZC12]

- **Extraction and Physical Verification:** Validates the layout for electrical correctness, logical consistency, and manufacturability. Includes Design Rule Checking (DRC), Layout Versus Schematic verification (LVS), Exclusive-OR comparison (XOR), antenna effect checks, and Electrical Rule Checking (ERC)—collectively known as sign-off checks before tape-out [KLMH11, KC24].

1.3 Next-Generation Lithography for IC Fabrication

Multiple next-generation lithography (NGL) methods have been introduced to support aggressive technology scaling. Prominent methods include Extreme Ultraviolet Lithography (EUVL), Nanowires, Electron Beam Lithography (EBL), and Multiple Electron Beam Lithography (MEBL), etc. These techniques can achieve feature sizes of 20 nm or below but introduce new manufacturing challenges, such as pattern distortions, that must be mitigated to ensure high-quality patterning. This section provides a brief description of different NGLs, whereas Section 2.4 gives a detailed description.

1.3.1 Extreme Ultraviolet Lithography

Extreme Ultraviolet Lithography (EUVL) [Lev22] is a leading next-generation lithography technique that employs 13.5 nm wavelength light to overcome sub-wavelength

limitations. However, this wavelength is largely absorbed by materials. Therefore, *clear-field masks* are used, which are coated with reflective optical components. Absorbers print the layout patterns, and light reflects from the vacant regions. Surface roughness in the reflective coating causes scattering, known as *flare* [FC12] as shown in Figure 2.6, which reduces contrast between dark (patterned) and bright (vacant) regions, degrading critical dimension (CD) uniformity [LC14]. EUVL uses 13.5 nm light and reflective clear-field masks to overcome sub-wavelength limits. Flare caused by reflective surface scattering reduces critical dimension (CD) uniformity, particularly in central chip regions. Flare is modeled using a PSF with vacancy-density maps, and minimizing flare is essential to preserve layout fidelity.

1.3.2 Nanowire

Silicon nanowires (SiNWs) have attracted significant research interest due to their superior physical properties and potential as active materials in future electronic and optoelectronic devices [RSV11]. It shows promise for future electronic and optoelectronic devices due to their excellent physical properties. Fabrication approaches include bottom-up and top-down techniques, though order and orientation remain difficult. Directed Self-Assembly (DSA) improves nanowire manufacturability by enabling controlled and ordered structures [SC15].

1.3.3 Electron Beam Lithography

It is a maskless technique for IC fabrication. It enables high-precision direct patterning with sub-10 nm resolution but suffers from low throughput and pattern distortions like proximity, forward, and backward scattering effects [PBM00]. The Multi-Column Cell (MCC) approach enhances throughput by using stencils and predefined characters, though stencil limitations still constrain efficiency [PBM00, YYGP13].

1.3.4 Multiple Electron Beam Lithography

Multiple Electron Beam Lithography (MEBL) is an emerging Next-Generation Lithography technique. MEBL employs thousands of e-beams in parallel to overcome EBL's

throughput limitations. Layouts are split into vertical stripes, but beam deflections at stitch-lines cause overlay errors, leading to violations such as via, vertical routing, and short polygon causing distortions. To ensure reliable fabrication, routing algorithms must incorporate lithography-aware constraints alongside standard design metrics [LFC15, HF20].

1.4 Motivation and Scope of this thesis

A key focus of this thesis is the exploration of techniques aimed at improving routing quality in the VLSI physical design flow. Routing is one of the most critical stages, involving combinatorial optimization problems that are mostly NP-hard, to determine precise interconnection paths, which directly impact signal integrity, propagation delay, power consumption, and overall chip performance.

Among emerging technologies, MEBL has gained attention as a powerful Next -Generation Lithography solution. To meet the unique challenges of MEBL, this thesis develops a dedicated global router tailored for MEBL, with Section 1.4.1.1 outlining the motivation for this direction. To further enhance routing quality under MEBL, we also design a specialized detailed router, with Section 1.4.1.2 providing the motivation behind its development. Together, these efforts aim to provide a comprehensive routing framework optimized for MEBL.

Apart from lithography-specific routing, a fundamental task that underpins nearly all global routing algorithms is the construction of Rectilinear Steiner Minimum Tree (RSMTs) [CW07, PMBSK23, TLCX20]. For a given set of n points (pins of a net), an RSMT is the shortest interconnection network in the 2D plane, potentially augmented with additional Steiner points. The *degree* of a net is defined as the number of pins present in a net. An RSMT minimizes the total wirelength under the L_1 (Manhattan) metric and serves as a backbone for many state-of-the-art global routers [CP06, CLYP09, Mof08, XZC09, WDL09, CLG$^+$10]. An RSMT of a 10-degree net is shown in Figure 1.5.

Since the RSMT construction is itself NP-hard [Chu04], its quality significantly influences the overall performance of the routing phase. To tackle this challenge, we introduce machine learning-based methods for both single-net RSMT construc-

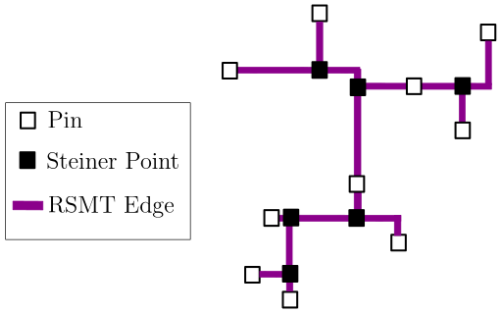


Figure 1.5: Example RSMT of a 10-degree net

tion and concurrent RSMT construction across multiple nets. Section 1.4.2.1 and Section 1.4.2.2 highlight the motivation and necessity of these approaches.

1.4.1 Routing Challenges in Multiple E-Beam Lithography

As discussed in Section 2.4.4, MEBL is a maskless direct-write process that overcomes the limitations of 193 nm immersion lithography. To achieve significantly higher throughput than conventional EBL, it uses thousands to millions of electron beams in parallel. These advantages make it a promising Next-Generation Lithography (NGL) technique. However, routing in MEBL remains challenging due to overlay errors arising from vias, vertical routing, and short-polygon violations described in Section 2.4.4. To address these issues, we propose dedicated global and detailed routers for MEBL, designed specifically to minimize these violations and improve routing quality.

The DP-based placement method in [LYZ+17] attempts to address stitch-line issues during placement by refining local congestion. However, since it operates before routing, its scope is restricted; it primarily mitigates violations due to fixed pin locations but cannot eliminate stitch-line violations that arise during subsequent routing stages. The method in [LFC15] augments a conventional global router with an additional congestion constraint, which reduces short-polygon violations in the track assignment and detailed routing but simultaneously increases via violations, creating an unfavorable trade-off between violation types.

The perturbation-based post-routing approach in [PBSK17] mitigates stitch-line violations through localized adjustments. However, it is unable to reduce vertical

routing violations significantly as it takes a pre-routed layout, making it unsuitable for congested designs. Similarly, the smart-boundary technique in [HF20] provides finer stitch control by adaptively defining routing boundaries, but its reliance on complex e-beam control limits scalability and practical deployment.

1.4.1.1 Design of Stitch-avoiding Global Router for MEBL

To the best of our knowledge, no dedicated global router explicitly accounts for stitch-lines during global routing. As detailed routing heavily depends on a global routing solution, it is essential to develop a global router tailored for MEBL to ensure stitch-line free detailed routing.

We considered mitigating the pattern distortions caused by the three violations in the global routing phase. Therefore, for MEBL, we have proposed a stitch-avoiding global routing technique to consider the presence of stitch-lines during global routing such that even a traditional detailed router can generate a routing result containing a minimum count of stitch-line violations. The proposed approach integrates dedicated net-shifting strategies, integer linear programming (ILP), and maze routing with several constraints and weights for stitch-line constraint to minimize stitch-line violations during global routing. By explicitly accounting for stitch constraints during global routing, our method significantly reduces the count of stitch-line induced violations compared to conventional global routing techniques.

1.4.1.2 Design of Stitch-avoiding Detailed Router for MEBL

Existing methods address violations at different design stages but lack an effective detailed routing solution. Incorporating stitch-line constraints into detailed routing enables simultaneous mitigation of vertical routing, via, and short-polygon violations, yielding a more practical and manufacturable approach for advanced MEBL layouts.

Therefore, for MEBL, we have proposed a Stitch-avoiding Detailed Router. The inputs to our detailed router are a 2D global routing solution, the location of stitch-lines, etc. Our method performs stitch-avoiding detailed routing to minimize stitch-line violations by categorizing nets based on their characteristics and using a multi-step routing strategy.

1.4.2 Challenges in RSMT Construction and Machine Learning-Based Solutions

The inherent difficulty in computing an optimal RSMT for a net arises from its NP-hard nature [Har82]. Hence, there are existing techniques that either generate optimal RSMTs in exponential time [JWWZ18] or in sub-exponential time [FLK+20] or non-optimal RSMTs in polynomial time using either heuristics [CW07] or machine learning methods [LCY21]. The algorithm in [LK19] builds the *ARSMT DB*, which enumerates all possible RSMTs for a given net on the Hanan grid.

`REST` [LCY21] employs reinforcement learning to construct RSMTs, achieving $\leq 0.36\%$ wirelength error for nets up to degree 50. Its progressive training scheme, where the degree-n model depends on a pre-trained degree-$(n-1)$ model, causes training time to scale poorly; for example, the increase from degree 50 to 51 is about $10\times$ larger than that from degree 10 to 11. Although effective in accuracy, such scalability limitations indicate that ML-based methods also face challenges, requiring further research to address them. However, the RSMT construction problem is not a standard ML task such as classification, regression, or clustering; rather, it is a complex optimization problem. Applying machine learning to this problem is non-trivial and requires specialized formulations.

1.4.2.1 Design of Single-net RSMT Construction Technique

The Hanan grid corresponding to a set of pin locations can be regarded as the input for constructing an RSMT. Since an RSMT on the Hanan grid exhibits a distinct graphical structure, it can naturally be visualized as an image. This motivates viewing RSMT construction as an *image-to-image translation* problem.

Recent advances in image-to-image conversion using Generative Adversarial Networks (GANs) motivate us to use a conditional GAN (cGAN)-based model for RSMT construction. Unlike standard GANs that generate images from noise, our task encodes the Hanan grid and pin locations as conditional inputs, enabling the generator to produce valid RSMTs with minimum wirelength. Using `GeoSteiner` [JWWZ18] solutions as supervised training data, we demonstrate the first successful application of image-to-image translation for RSMT construction.

1.4.2.2 Design of Concurrent RSMT Construction Across Multiple Nets

All existing state-of-the-art methods construct RSMTs sequentially, processing one net at a time. As modern ICs contain a large number of nets, simultaneous construction of RSMTs for multiple nets can potentially decrease the total time required to build all nets in an IC, and during concurrent construction, the edge overlapping between RSMTs of different nets can also be reduced.

As constructing an optimal RSMT for a single net remains sub-exponential at best [FLK+20], this motivates us to adopt a machine-learning-based approach to concurrently generate RSMTs for multiple nets in polynomial time while minimizing total wirelength error, time, and group-wise edge overlap. For multiple nets, we first generate the individual Hanan grids and take their union to form a combined grid containing all pins. To distinguish pins and edges across nets, nodes and edges must include features indicating their net membership; some edges may belong to multiple RSMTs, so their features must encode multi-net associations. This unified grid enables concurrent RSMT construction.

A Graph Neural Network (GNN) can process these node and edge features to predict RSMTs across multiple nets. GNNs operate on graph-structured data and capture dependencies through node connectivity, making them suitable for tasks such as node classification and link prediction. As a supervised method, the GNN requires labeled graph data. The combined Hanan grid serves as the input graph, and ground-truth labels can be derived using `GeoSteiner` [JWWZ18] to compute optimal RSMTs for each net, followed by overlap mitigation and union of the resulting RSMTs. Edges in the combined grid can then be labeled accordingly. This motivates the use of a GNN for concurrent RSMT construction for multiple nets that reduces overall generation time, minimizes group-wise edge overlaps, and achieves minimum wirelength.

1.5 Organization of the Thesis

The subsequent chapters of this thesis are organized as follows:

- Chapter 1 covers the basic introduction of VLSI design flow, physical design flow, Next-generation lithography techniques, followed by the scope of the thesis.

- Chapter 2 reviews related work, covering traditional global and detailed routing techniques, placement and routing methods in MEBL and RSMT construction methods, and recent machine learning approaches for EDA applications.
- Chapter 3 presents our proposed stitch-avoiding global routing for MEBL.
- Chapter 4 describes our proposed method for stitch-avoiding detailed routing for MEBL.
- Chapter 5 proposes the construction of RSMT with a conditional generative adversarial network.
- Chapter 6 focuses on our proposed method for concurrent construction of RSMTs of multiple nets with a Graph Neural Network.
- Chapter 7 provides a summary of the contributions of this thesis and identifies promising paths for future research.

CHAPTER

TWO

BACKGROUND AND RELATED WORKS

Contents

2.1 Introduction **20**

2.2 Physical Design Techniques for Photolithography **21**

2.2.1 Partitioning 21

2.2.2 Floorplanning 23

2.2.3 Placement 23

2.2.4 Routing 24

2.2.5 Extraction and Physical Verification 29

2.3 Photolithography: Enhancements and Constraints **30**

2.3.1 Optical Proximity Correction 30

2.3.2 Multiple Patterning 30

2.3.3 Limitations of OPC and MP 33

2.4 Next Generation Lithography Techniques and Limitations **34**

2.4.1 Extreme Ultraviolet Lithography 34

2.4.2 Nanowire 36
2.4.3 Electron Beam Lithography 37
2.4.4 Multiple Electron Beam Lithography 38
2.5 Existing Approaches to Mitigate Stitch-Line Violations **40**
2.5.1 Comparative Analysis of Existing MEBL-Aware Methodologies . . 44
2.5.2 Research Gap and Motivation 46
2.6 Baseline Routing Framework **46**
2.6.1 BoxRouter: A Traditional Global Router 47
2.6.2 RegularRoute: A Traditional Detailed Router 49
2.7 Machine Learning for Physical Design Automation **51**
2.7.1 ML for Floorplanning and Placement Heuristics 52
2.7.2 ML for Global Placement and Routability Optimization 52
2.7.3 ML for Routing, Routing Congestion and Routability Prediction . 53
2.7.4 ML for Crosstalk and Signal Integrity Prediction 55
2.7.5 ML Methods in Lithography Hotspot Detection 55
2.7.6 Categories of Machine Learning Methods for Physical Design Problems 57
2.8 Existing Approaches for RSMT Construction **59**
2.8.1 Comparative Analysis of Existing RSMT Construction Methodologies 62
2.8.2 Research Gap and Motivation 62
2.9 Summary **62**

2.1 Introduction

This chapter begins with a comprehensive overview of physical design techniques for photolithography in Section 2.2. We then discuss various photolithography en-

hancement methods in Section 2.3, highlighting their advantages as well as inherent limitations. Section 2.4 introduces next-generation lithography (NGL) techniques, providing a detailed discussion of their capabilities and constraints. In Section 2.5, we review the existing methods for minimizing stitch-line induced violations in MEBL, critically analyzing their shortcomings. Section 2.6 briefly describes traditional global and detailed routers called BoxRouter [CP06] and RegularRoute [ZC12] respectively. Section 2.7 offers an in-depth survey of machine learning applications in Physical Design Automation within Electronic Design Automation (EDA), summarizing state-of-the-art approaches and research trends. Section 2.8 presents a review of the literature on RSMT construction methods, along with key properties relevant to VLSI design. Finally, Section 2.9, gives the chapter summary.

2.2 Physical Design Techniques for Photolithography

This section provides a brief description of all the phases of VLSI physical design introduced in Section 1.2.

2.2.1 Partitioning

Partitioning decomposes large systems with millions or billions of transistors into smaller, manageable modules or blocks [KLMH11, She99]. Each block is defined by a *netlist*, where a net specifies terminals to be electrically connected. Figure 2.1(a) shows a circuit and Figure 2.1(b) its division into three subcircuits with four interconnections.

Formally, the problem is modeled as a *hypergraph* $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ are components and $E = \{e_1, \ldots, e_m\}$ are *hyperedges* (nets), each $e_i \subseteq V$ denoting interconnected components [AK94]. The area of component v_i is $a(v_i)$, and partitioning divides V into k disjoint subsets $V_1, \ldots, V_k$ such that $\bigcup_{i=1}^{k} V_i = V$ and $V_i \cap V_j = \varnothing$ for $i \neq j$.

Partitioning is commonly described as a *cut*, whose cost is quantified by the *cut-size*, meaning the total number of hyperedges that span across different partitions [KL70, FM82]. Let c_{ij} represent the cut-size between partitions V_i and V_j. For

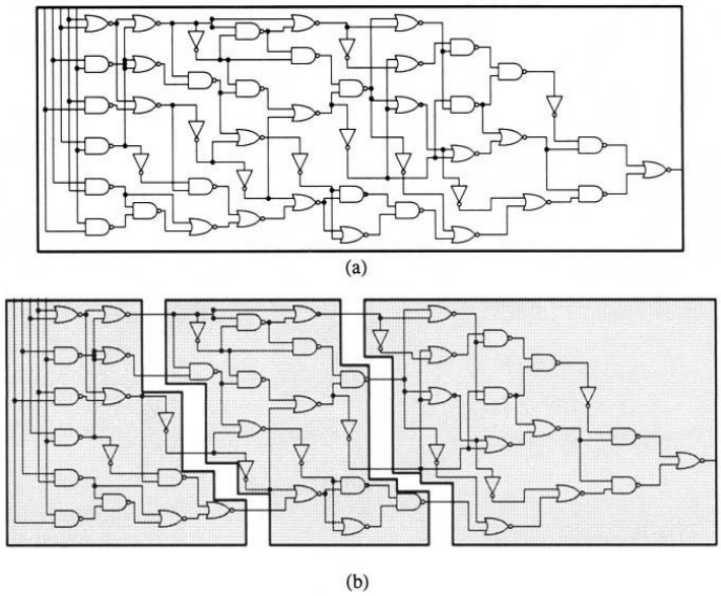


Figure 2.1: Partitioning of a Circuit (Courtesy [She99])

any partition V_i, its area is given by $Area(V_i) = \sum_{v \in V_i} a(v)$, and its terminal count is $Count(V_i)$, both constrained by $A_i^{\min} \leq Area(V_i) \leq A_i^{\max}$ and $Count(V_i) \leq T_i$. Furthermore, the total number of partitions must satisfy $K_{\min} \leq k \leq K_{\max}$ [GHM+24]. For hyperpaths $P = \{p_1, \ldots, p_m\}$, with $H(p_i)$ denoting the number of cuts on p_i, the objectives are: minimize inter-partition connections (mincut) [HK92], minimize delay by reducing $\max_{p_i \in P} H(p_i)$ [AK94], and satisfy terminal and area constraints.

Recently, machine learning—especially Graph Neural Networks (GNNs) and hybrid representation learning—has been applied to augment heuristics. TP-GNN learns node embeddings for 3D IC tier partitioning with timing, hierarchy, and technology awareness [LPZ+20]. VLSI Hypergraph Partitioning with Deep Learning explores learning-based strategies for netlist hypergraphs [KODG24]. K-SpecPart combines supervised embeddings with multilevel partitioning, outperforming hMETIS [MHR06] and KaHyPar [BKK+22]. Spectral coarsening approaches such as SHyPar use structural metrics like effective resistance for improved coarsening [SAF25]. Together, these works highlight the potential of integrating heuristics with learned representations for higher-quality partitioning.

2.2.2 Floorplanning

After circuit partitioning, as introduced in Section 1.2, this step first estimates the approximate area of all the blocks (sub-circuits), determines feasible block shapes, and specifies the number of pins [KLMH11, She99]. The netlist defining inter-block connections is also available. Completing the layout requires assigning concrete shapes to blocks, arranging them on the layout, and depending on the netlist, interconnecting their pins accordingly. This arrangement proceeds in two stages: *floorplanning*, which determines block dimensions and interconnect planning [CCWW00], and *placement*, which assigns exact locations to blocks [VNA+07].

The circuit blocks $B_1, B_2, \ldots, B_n$ with areas $a_1, a_2, \ldots, a_n$ are the inputs to this step. Each block B_i has aspect ratio bounds A_i^l and A_i^h, specifying minimum and maximum allowable aspect ratios. In this step, the block dimension is determined by the floorplanning algorithm so that Equation 2.1 holds while minimizing total layout area [AM04] where h_i and w_i denote the height and width of block B_i respectively.

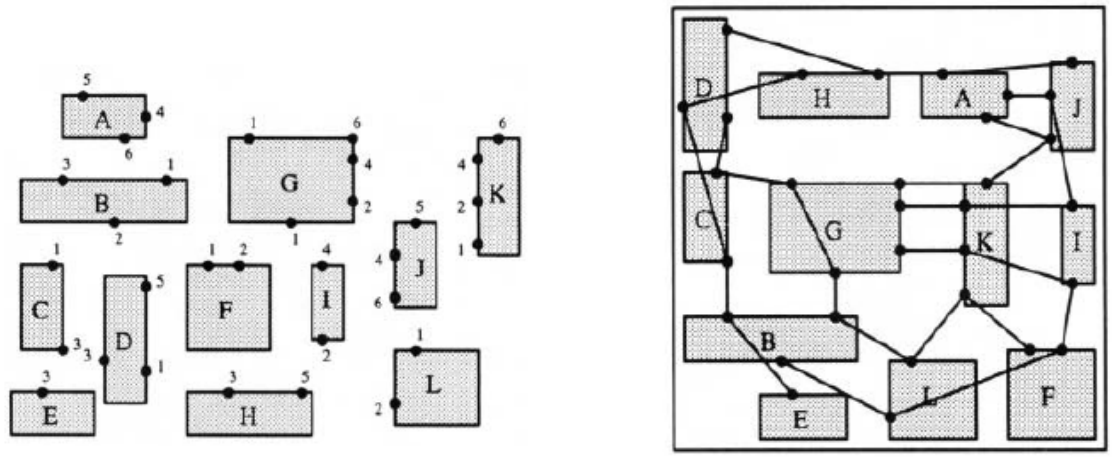
$$A_i^l \leq \frac{h_i}{w_i} \leq A_i^h \tag{2.1}$$

2.2.3 Placement

This phase determines the coordinates of circuit blocks on the layout surface as introduced in Section 1.2. Given a list of fixed blocks with dimensions and a netlist containing nets with details of the pins on block boundaries and on the periphery, the objective is to produce a routable layout with minimal area [KLMH11, She99]. For high-performance designs, delay minimization is also critical, typically approximated by reducing the longest net length, yielding *timing-driven* placement methods [XR05]. Placement quality is measured by area, routability, and performance.

Let $B = \{B_1, \ldots, B_n\}$ denote circuit blocks with dimensions (w_i, h_i), and the set of nets $N = \{N_1, \ldots, N_m\}$. The task is to assign non-overlapping rectangles $R = \{R_1, \ldots, R_n\}$ to blocks such that the bounding area and total wirelength $\sum_i L_i$ are minimized, while ensuring routability. For timing-driven placement, the maximum net length $\max_i L_i$ is minimized.

The placement involves NP-hard [MHK12] problems, motivating heuristic and



(a) Blocks and interconnecting nets (b) Placement solution

Figure 2.2: Block placement (Courtesy [She99])

ML-based approaches. Classical methods include min-cut [HK92], force-directed [VPC07], and quadratic / analytical placers such as mPL [CCS+06] and NTUplace [CJH+07]. More recent advances include RePlAce, an open-source analytic placer with routability optimization [CKKW18]. DREAMPlace, a GPU-accelerated framework that uses deep learning toolkits [LDL+19, LPRK20, GJLP20, LGG+23] and ML-based approaches such as the method in [MGY+21], uses reinforcement learning to place macros. Multi-terminal nets are modeled using Rectilinear Steiner Minimum Trees (RSMTs) [LCY21], although exact RSMT construction is NP-hard [Han66].

2.2.4 Routing

The routing phase aims to complete all netlist interconnections with minimal wirelength [She99, KLMH11] as highlighted in Section 1.2. It involves multiple sub-steps: *global routing*, *layer assignment*, *track assignment*, and *detailed routing* [CLG+10]. Global routing assigns approximate paths across regions, while layer and track assignment specify routing layers and tracks, sometimes integrated into detailed routing [KSKM18]. Detailed routing finalizes exact pin-to-pin connections, including wire

geometry, spacing, and layer usage. Since routing is computationally hard, heuristic methods are widely used [CP06, CW07], while in recent years, machine learning approaches have also gained attention [LZD+24, CSD+24, CC09a]. Since complete routing is not always feasible, *rip-up and re-route* [PXZC12] methods are often applied to resolve congestion. Section 2.2.4.1, 2.2.4.2, 2.2.4.3, and 2.2.4.4 outline each sub-step.

The exact coordinates of each circuit block with pin locations are determined during placement, and a netlist specifying the required interconnections is also generated [CJH+07, CKKW18, LDL+19, MGY+21]. The unoccupied space forms *routing regions*, within which interconnects must be established. For the routing problem, the following are the required inputs:

- Netlist: Set of nets and their required interconnections between circuit components.
- Timing constraints: Timing budgets for nets, typically applied to critical nets to meet performance requirements.
- Placement information: Exact locations of circuit blocks, pin positions on block boundaries, and I/O pin locations on the chip periphery.
- Routing regions: Unoccupied chip area available for interconnects, organized across multiple routing layers.
- Interconnect delay parameters: For each metal layer, per unit length resistance and capacitance (RC) delay, including RC delay for vias connecting wire segments across layers.

2.2.4.1 Global Routing

This is commonly formulated by using graphs, where regions for routing with their connectivity and routing capacities are represented as a graph [She99, KRW05]. The choice of graph model depends on the design style, leading to multiple different graph models [She99, CC09a]. Among these, the grid graph is the simplest and most widely used model for routing.

A layout is represented as the grid graph $G = (V, E)$ (Figure 2.3). The layout is divided into $h \times w$ zones of unit square cells, where each cell c_i corresponds to a

vertex $v_i \in V_1$. An edge $(v_i, v_j) \in E_1$ exists if the corresponding cells are adjacent. All terminals of c_i are mapped to their corresponding vertex v_i. Each edge is assigned unit capacity $c(e) = 1$ and unit length $l(e) = 1$. Blocked cells are modeled by setting the capacities of all incident edges to zero. Given a grid graph, the global routing problem reduces to finding a connecting global routing path between terminal vertices for a two-terminal net, and for multi-pin nets, finding an RSMT [Han66, CSD+24]. The problem of global routing is formally defined as: Given,

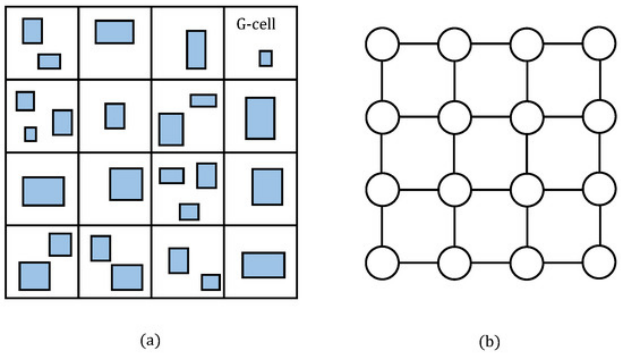


Figure 2.3: Grid Graph Model (Courtesy [She99])

- a netlist $N = \{N_1, N_2, \ldots, N_n\}$
- a grid graph $G = (V, E)$

the task is to construct RSMT T_i for every N_i $(1 \leq i \leq n)$ that satisfies the edge-capacity constraints [She99, KRW05, KSKM18]. Specifically, for every edge $e_j \in E$,

$$U(e_j) = \sum_{i=1}^{n} x_{ij} \leq c(e_j) \tag{2.2}$$

where, if $e_j \in T_i$ then $x_{ij} = 1$ and otherwise it is 0. The major aim is to minimize the total wirelength [Han66, She99, PXZC12, LZD+24],

$$\min \sum_{i=1}^{n} L(T_i) \tag{2.3}$$

Our proposed stitch-avoiding global routing for MEBL took inspiration from the

conventional global router called BoxRouter [CP06] where a progressive ILP is used to limit the size of the ILP. Section 2.6 provides a brief description of BoxRouter. Recent global routing advances leverage machine learning and graph-based optimization to improve wirelength, congestion, and timing. Neural network-guided routers learn congestion-aware patterns to reduce wirelength and hotspots [LZD+24, CSD+24]. Reinforcement learning sequentially constructs Steiner trees for multi-terminal nets, balancing capacity and delay [LLH+24]. Hybrid methods combine classical heuristics, such as maze routing or ILP, with learned predictions to guide search and resource allocation [LCZ22, PXZC12], achieving notable improvements in routability and timing closure for large-scale VLSI designs.

2.2.4.2 Layer Assignment

Layer assignment is a critical step after global routing [LW09, DAL15]. Given a set of nets represented as RSMTs and the number of available metal layers specified by the foundry, layer assignment assigns vertical and horizontal segments of every net to a specific routing layer. The objective is to maximize routability while minimizing routing congestion, via count, delay, and total wirelength [DAL15]. The output is a layer-mapped routing solution that is manufacturable and optimized for performance. Hence, it can be formally defined as: Given a global routing solution of each net $\mathcal{N} = \{N_1, \ldots, N_n\}$, routing graph $G = (V, E)$ with K metal layers, per-layer capacities $c_{e,k}$, wirelength w_k, and via delay w^{via}. Define $y_{s,k} = 1$ if segment s is on layer k, else 0 and $z_{s,s'} = 1$ if via connects s and s', else 0. The objective is to minimize congestion, total wirelength, and via count as shown in Equation 2.4, where $U(e, k)$ is the number of segments using edge e on layer k.

2.2.4.3 Track Assignment

This is one of the intermediate steps between global and detailed routing. In this step, wire segments are assigned to horizontal or vertical routing tracks defined by the foundry [CC09a, KSKM18]. Tracks are separated by a *spacing*, and the distance between adjacent tracks is referred to as the *pitch* [PXZC12, LZD+24]. All metal features must maintain a minimum separation equal to the spacing. Proper track

$$\min \sum_{k,e} \max(0, U(e,k) - c_{e,k}) + \sum_{s,k} w_k \ell(s) y_{s,k} + \sum_{s,s'} w^{\text{via}} z_{s,s'} \tag{2.4}$$

subject to:

$$\sum_{k=1}^{K} y_{s,k} = 1, \qquad \forall s \tag{2.5}$$

$$U(e,k) \leq c_{e,k}, \qquad \forall e,k \tag{2.6}$$

$$z_{s,s'} \geq |y_{s,k} - y_{s',k}|, \qquad \forall (s,s'),k \tag{2.7}$$

$$y_{s,k}, z_{s,s'} \in \{0,1\}. \tag{2.8}$$

assignment enables optimization of signal delay, reduction of cross-talk, and adherence to process design rules [CP06, LCZ22].

Several methods have been proposed for track assignment. Left-edge assignment sequentially places wires using the leftmost available track, providing simple and efficient solutions for two-terminal nets [Yos84]. Integer linear programming (ILP) approaches optimize track usage globally, minimizing conflicts and wirelength for complex multi-terminal nets [KSKM18]. Graph-based heuristic methods leverage conflict graphs to assign tracks while considering congestion and signal integrity [CP06, LZD+24]. Recent work also explores machine learning-guided assignment, predicting optimal track allocation patterns from routing congestion maps [LCZ22].

2.2.4.4 Detailed Routing

Detailed routing is the final step of the routing process, completing all netlist connections. During global routing, approximate paths are obtained as a subset of global routing regions for each net without specifying exact geometries [She99, CC09a]. Layer and track assignments ensure wire segments are placed on appropriate metal layers and tracks while satisfying design rules and minimizing wirelength and vias [KSKM18, ZC12]. Detailed routing then assigns rectilinear wire segments within these regions, completing interconnections; in some approaches, layer and track assignment are integrated into detailed routing [LZD+24].

Let $\mathcal{N} = \{N_1, \ldots, N_n\}$ denote the netlist with terminals $\mathcal{T}_i = \{t_1^i, \ldots, t_{k_i}^i\}$. For each net N_i, the global router constructs a path P_i through routing regions $\mathcal{R} \subseteq G$, where G is the routing grid. Detailed routing assigns rectilinear wire segments $S_i = \{s_1^i, \ldots, s_{m_i}^i\} \subseteq P_i$ such that:

- Each segment s_j^i connects points within P_i or adjacent segments.
- Segments follow spacing rules: $\forall s_j^i, s_k^i$ on the same layer, $dist(s_j^i, s_k^i) \geq$ spacing.
- All terminals are connected: $\forall t \in \mathcal{T}_i,\ \exists\ s_j^i$ incident on t.
- Total wirelength $L_i = \sum_{j=1}^{m_i} \ell(s_j^i)$ and vias v_i are minimized while respecting layer assignments.

Formally, the detailed routing problem is:

$$\min \sum_{i=1}^{n} \left(\sum_{s_j^i \in S_i} \ell(s_j^i) + \sum_{v \in V_i} w^{\text{via}} \right), \quad \text{s.t. } S_i \subseteq P_i,\ \forall i = 1, \ldots, n, \tag{2.9}$$

where V_i is the set of vias for N_i with cost w^{via}. Detailed routing ensures all nets are connected while satisfying spacing, layer, and design rules [ZC12, PXZC12, LZD+24]. Our stitch-avoiding detailed routing for MEBL has taken inspiration from a conventional detailed router called RegularRoute [ZC12], briefly described in Section 2.6.2.

2.2.5 Extraction and Physical Verification

Physical verification ensures correct electrical functionality, logical equivalence, and manufacturability using EDA tools as introduced in Section 1.2. Key checks include Electrical Rule Check (ERC), Layout versus Schematic (LVS), Design Rule Check (DRC), and antenna checks, etc. [KLMH11, KC24]. Collectively, these *sign-off checks* must be passed before tapeout [KC24].

DRC enforces foundry geometrical constraints [KLMH11]; LVS compares the extracted layout netlist with the schematic [KLMH11, KC24]; XOR detects unintended layout changes after ECOs [KC24]; antenna checks identify charge accumulation risks on interconnects [KLMH11, KC24]; and ERC validates power/ground connectivity, floating nets, and rule compliance [KC24].

2.3 Photolithography: Enhancements and Constraints

Few techniques have been developed to mitigate the sub-wavelength lithography gap problem, described in Section 1.1 and Figure 1.3, to extend immersion lithography. As introduced in Section 1.1, Optical Proximity Correction (OPC) [CLC08] and multiple patterning [YRGP15] are two widely used approaches. In OPC, additional features are added to the layout to minimize pattern distortion [CLC08]. Multiple patterning decomposes the layout into multiple masks, enabling continued scaling despite the wavelength limitations of exposure light [YYDP15, YRGP15, SRM+22].

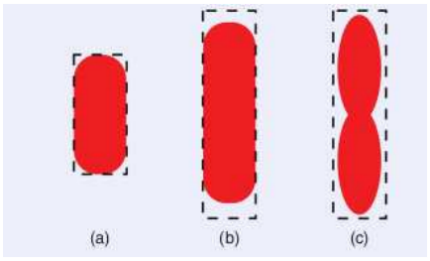
2.3.1 Optical Proximity Correction

It is a popular resolution-enhancement technique (RET) that modifies mask patterns by adding features to compensate for distortions [CC09b]. Major distortions such as line-end shortening, corner rounding, and line-width shrinking, as shown in Figure 2.4a, are mitigated using hammerheads, serifs, and line biasing, as shown in Figure 2.4b. Figure 2.4c shows the improvement achieved with OPC [CLC10].

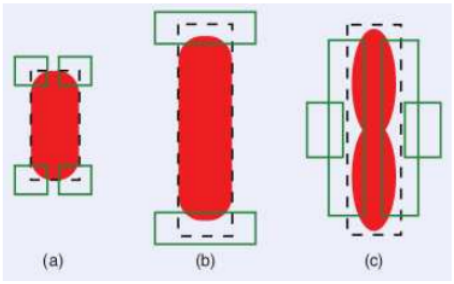
Several works extend the OPC. Forte et al. [FS12] proposed PUFs-aware OPC, targeting IC regions that contain physically unclonable functions (PUFs), which are particularly distortion-sensitive. Huang et al. [HLC10] developed an efficient OPC model using interfered intensity evaluation, applying wire-segment perturbation to maximize interference slack. Chen et al. [CLC08] introduced OPC-aware routing, generating demand maps with quasi-inverse lithography to minimize conflicts between OPC features and existing wires as shown in Figure 2.4d. In general, OPC remains essential for printing sub-wavelength ICs (e.g., 45 nm and above with 193 nm immersion lithography) [PYC+12].

2.3.2 Multiple Patterning

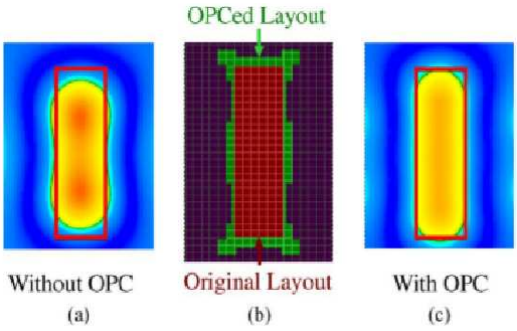
The multiple patterning (MP) technique is developed towards the continuation of IC fabrication using 193 *nm* wavelength. MP has different variations depending on the number of masks it uses to decompose the layout. For example, two masks are used in double patterning lithography (DPL), three masks are used in triple patterning



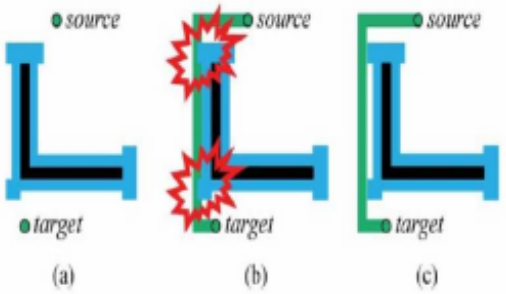
(a) Major pattern distortions (I) Corner rounding. (II) Line-end shortening. (III) Line-width shrinking. (Courtesy [CC09b])



(b) Mitigation techniques (I) Serif. (II) Hammerhead. (III) Line biasing. (Courtesy [CC09b])



(c) Printing layout with OPC compensation. (Courtesy [CLC10])

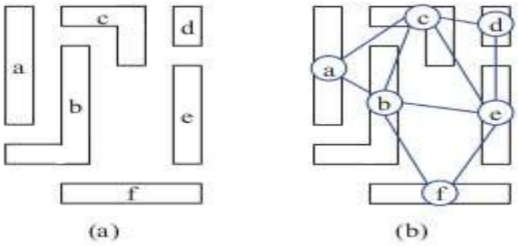


(d) (I) Layout with OPC prohibited regions. (II) Conflict. (III) OPC-aware routing (Courtesy [CLC10])

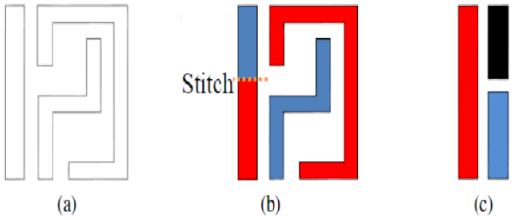
Figure 2.4: Major pattern distortions and improvements with OPC

lithography (TPL), and so on. However, in the works [YYDP15, YP14], a patterning distance d greater than the size of the feature is defined, and two patterns placed on the same mask with a distance less than d between them are called conflicts.

A conflict graph C(V, E) is defined in the state-of-the-art research to facilitate the decomposition. In a conflict graph, each pattern is considered as a vertex $v \in V$, and an edge (v_i, v_j) exists if patterns i and j cause a conflict. Figure 2.5a shows a layout along with its associated conflict graph, as presented in [FCC12a]. Conflict minimization can lead to the decomposition of a single pattern into multiple masks, causing stitches as shown in Figure 2.5b. If a pattern has on the stitch, then two divisions of it along the stitch get printed in different passes. This situation leads



(a) (a) Layout, (b) Conflict graph (Courtesy [FCC12a])



(b) (a) Layout, (b) Two masks are used, different part of the same pattern gets printed using different masks, causing distortion (c) An increasing number of masks is the only solution. (Courtesy [CLC10])

Figure 2.5: Multiple Patterning

to distortion during fabrication. Hence, an important objective is to minimize the number of stitches.

Various works have been proposed on multiple patterning. The method in [FCC12b] developed a stitch-aware wire perturbation technique for double patterning lithography. In conflict graphs, odd cycles indicate native conflicts, which cannot be resolved by stitch insertion. Their geometry-based method predicts such conflicts and applies x/y-directional perturbation, reducing native conflicts by an average of 85%.

With aggressive feature-size scaling, double patterning reached its limits, leading to triple and quadruple patterning. Yu, Yuan et al. [YYDP15] proposed a triple patterning layout decomposition via graph coloring, where colors represent masks. The problem was formulated as an NP-hard ILP, whose runtime scales poorly with input size. To accelerate it, they introduced (1) *Layout Graph Simplification*, (2) *Independent Component Computation*, and (3) *Bridges Computation* [YYDP15]. These techniques reduce ILP problem size while preserving optimality. Since ILP remained slow, they proposed a novel vector programming approach, relaxed into semidefinite programming (SDP), reducing runtime to polynomial complexity. This method reduced stitches by 7% while introducing 9% more conflicts.

Fang et al. [FCC12a] also addressed triple patterning decomposition via graph coloring, an NP-complete problem. They applied graph reduction (Connected Component Computation, Vertex Degree ≤ 3 Removal, 2-Edge-Connected, 2-Connected,

and 3-Edge-Connected Component Computations), producing smaller subgraphs for coloring. A stitch-aware mask assignment was then applied, reducing conflicts by 56%. Yu and Pan extended this to quadruple patterning [YP14], combining a semidefinite algorithm with linear color assignment.

Yu et al. [YLLP+13] proposed a density-aware balanced triple patterning decomposition using semidefinite programming (SDP). By integrating density-awareness into graph simplification (independent component computation, vertex-degree less equal to 3 removal, and 2-edge/2-vertex-connected component computations), their method achieved more balanced layouts with reduced edge placement error, while maintaining comparable conflict and stitch counts.

More recent advances include GPU-accelerated exact cover / matrix-cover techniques for MPL layout decomposition by Chen et al. [CYY23], achieving over 10× speed-ups without loss in quality. In the SAT domain, Liu et al. [LLZ+24] transformed TPL decomposition into SAT-based coloring, offering competitive performance. Evolutionary approaches such as DEA-PPM have also been proposed for generalized MPL decomposition, balancing decomposition quality and runtime scalability [CXX23]. In routing-aware patterning, Mr.TPL offers a TPL-aware multi-pin net routing method, reducing color conflicts by 81.17 %, stitches by 76.89 %, and speeding up by up to 5.4× compared to prior TPL routing methods [WJK+25]. Finally, open-source frameworks such as OpenMPL provide modular platforms for implementing and benchmarking layout decomposition flows [LMS+20].

2.3.3 Limitations of OPC and MP

Optical Proximity Correction (OPC) and Multiple Patterning (MP) were introduced to extend the life of immersion lithography for advanced IC nodes. While effective for larger nodes, both techniques are now reaching fundamental and economic limits, driving the search for next-generation lithography (NGL).

OPC has been effective for feature sizes of 45 nm and above [PYC+12], but its ability to correct complex pattern distortions degrades significantly below 45 nm. The growing complexity of mask optimization leads to excessive computational cost and diminishing returns. For instance, deep learning-enabled OPC workflows (such as

DAMO) show promising speedups but still face issues of mask manufacturability and residual error [CCM+20].

Similarly, MP decomposes layouts into multiple masks to resolve pitch limitations. Although double and triple patterning has been widely studied, quadruple patterning has already been demonstrated [YYDP15]. However, each additional mask exponentially increases fabrication cost and process variability. At sub-7 nm nodes, the required mask count renders MP economically unsustainable [CYY23, WJK+25].

These limitations make OPC and MP stopgap solutions rather than long-term enablers, underscoring the importance of alternative approaches such as EUV lithography and complementary NGL technologies [Lev22, Nei21] introduced in Section 1.3.

2.4 Next Generation Lithography Techniques and Limitations

2.4.1 Extreme Ultraviolet Lithography

As introduced in Section 1.3.1, Extreme Ultraviolet Lithography (EUVL) is a leading next-generation lithography technique that employs 13.5 nm wavelength light to overcome sub-wavelength limitations. However, this wavelength is largely absorbed by materials. Therefore, *clear-field masks* are used, which are coated with reflective optical components. Layout patterns are printed using the absorbers, and light reflects from the vacant regions. Surface roughness in the reflective coating causes scattering, known as *flare* as shown in Figure 2.6, which reduces the contrast between dark (patterned) and bright (vacant) regions, degrading critical dimension (CD) uniformity [FC12].

The flare in EUVL systems is typically modeled using a Point Spread Function (PSF). The original intensity of the layout I_o is convolved with the PSF to generate a flare map. For each cell of a gridded layout, pattern density often approximates I_o within each cell to reduce computational overhead [SWT05, ZHW+11]. Since flare is generated on the clear-field masks primarily due to vacant regions, instead of pattern density, a vacancy density map is used. The layout is divided into grid

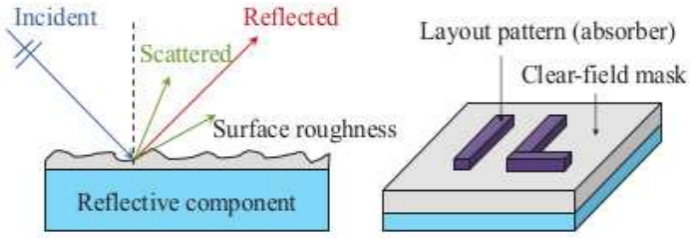


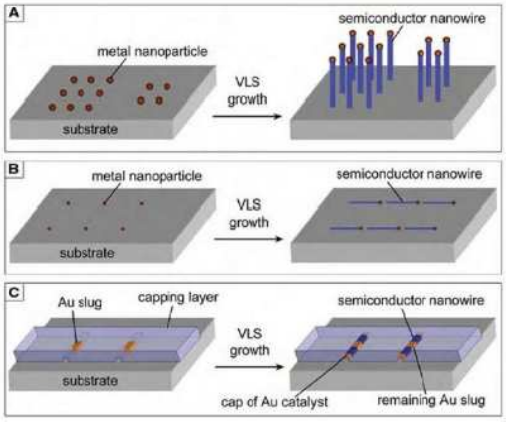
Figure 2.6: The photo-reflective layer on the mask redirects light, and the pattern is generated using sections with light-absorbing materials. (Courtesy [FC12])

cells of rectangular shape. For each cell (x, y), the flare $F(x, y)$ is calculated by Equation 2.10 where $D_v(x, y)$ and $PSF(x, y)$ are the vacancy density and PSF of the cell (x, y), respectively.

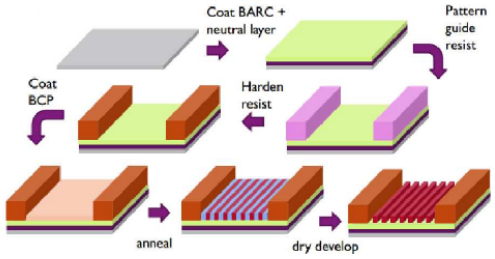
$$F(x, y) = D_v(x, y) \otimes PSF(x, y) \tag{2.10}$$

The central regions of a chip experience higher flare than the periphery, a phenomenon called the *flare periphery effect* [FC12]. The global flare distribution exhibits a concave profile [FC12, LC14]. State-of-the-art methods model the PSF using a 2D Gaussian function to capture this periphery-dependent flare behavior. The flare reduces pattern contrast, causing critical dimension (CD) variation and degraded layout fidelity. Therefore, minimizing flare is essential for accurate pattern printing.

State-of-the-art methods model the PSF using a 2D Gaussian function to capture this periphery-dependent flare behavior. Recent works has focused on rigorous flare modeling and stochastic effects in EUVL [PBSK21, Fuj22, DBH18]. the method in [PBSK21] shows that post-layout re-allocation of wire segments (i.e., altering wire placement after layout synthesis) can reduce flare variation by 20% and average flare by 11%, while also lowering the need for dummy fills by comparable amounts, without degrading layout performance. Methods such as simultaneous wire segment perturbation and selective dummification proposed in their work are particularly relevant for sub-7nm nodes [PBSK19]. Flare reduces pattern contrast, causing CD variation and degraded layout fidelity; therefore, minimizing flare remains essential for accurate pattern printing at sub-5nm nodes.



(a) Bottom-up approach for manufacturing of nanowire. (Courtesy [ZMCL17])



(b) Nanowire creation using Directed Self-Assembly (Courtesy [GRDY+12]).

Figure 2.7: Nanowire Lithography

2.4.2 Nanowire

Silicon nanowires (SiNWs), described in Section 1.3.2, have emerged as a promising research area owing to their exceptional physical properties and their potential application as active materials in future electronic and optoelectronic devices [RSV11]. Figure 2.7a illustrates the bottom-up fabrication approach, where nanowires grow from the substrate via chemical processes. Both bottom-up and top-down methods are widely used for fabricating horizontal and vertical nanowires. However, achieving precise orientation and order remains challenging. To address this, *Directed Self-Assembly* (DSA) employs template-based synthesis to fabricate one-dimensional nanowires with controlled topological geometry [SC15], as shown in Figure 2.7b.

Recent advancements have further enhanced SiNW fabrication techniques. For instance, the method in [HWL+23] demonstrated the fabrication of SiNWs using metal-assisted chemical etching combined with micro-vibration, improving the etching rate and uniformity. Additionally, a review by Yang provided an overview of the classification and self-assembly theories of anisotropic nano-building-blocks, with a particular focus on recent developments in SiNW fabrication [YC23]. These advancements contribute to the ongoing progress in SiNW technology, enabling more precise and scalable fabrication methods for future applications.

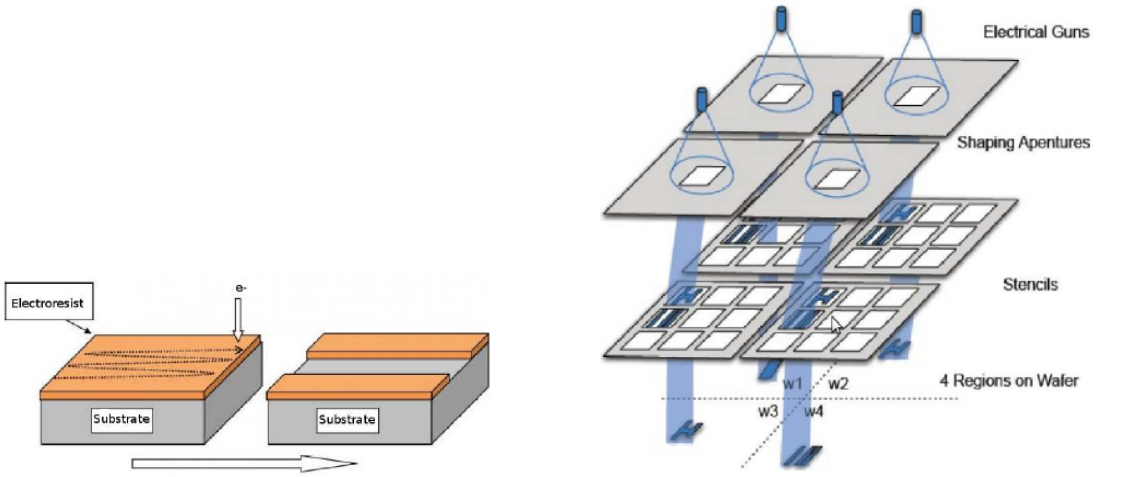
2.4.3 Electron Beam Lithography

As described in Section 1.3.3, Electron beam lithography (EBL) is a maskless technique for IC fabrication. It uses a resist, which is a surface covered with an electron-sensitive film. A focused beam of electrons scans the resist to draw custom shapes. E-beam changes the chemical structure of the resist, leading to the fabrication of the layout patterns. This form of maskless lithography is known as direct write.

The relationship among exposure area A, beam current I, exposure time T, and electron dose D is given by $D \cdot A = I \cdot T$ [PBM00]. In EBL, computation of the desired dose for a given area is very important regarding the mitigation of pattern distortion. Figure 2.8a illustrates the electron gun and scanning technique. EBL achieves sub-10 nm resolution, making it suitable for high-precision patterning. However, it suffers from low throughput since only a single electron beam is used. Other challenges are electron proximity effect, forward scattering, and backward scattering. The proximity effect arises from improper dose distribution, causing patterns wider than intended. Forward scattering occurs when secondary electrons deviate laterally within the resist, while backward scattering redirects primary electrons back toward their origin. These issues, combined with low throughput, limit the widespread adoption of EBL. To improve the throughput of EBL, the technique is expanded to support a Multi-Column Cell (MCC) framework, which operates in two modes: (1) Variable Shaped Beam (VSB) [SMM08] and (2) Character Projection (CP) [SMM08].

MCC decomposed layout patterns into rectangles termed *characters*, which can be pre-designed on stencils. If available on the stencil, a character can be printed in a single shot using CP, enabling parallelism across wafer regions and significantly improving throughput compared to pure VSB mode [YYGP13]. However, modern ICs contain an enormous variety of patterns, requiring excessive and costly stencils. Since only a limited number of characters can be designed on a stencil, non-predefined patterns must still be written using VSB. Thus, efficient stencil planning has become a critical research direction to balance throughput gains and stencil limitations.

To balance stencil cost and throughput, only the most frequently occurring patterns are pre-designed on stencils. Figure 2.8b shows an MCC layout where the wafer is divided into four regions, each printed using a separate CP, increasing throughput. The total writing time is determined by the region with the maximum CP time.



(a) Electron Beam Lithography (Courtesy [FS17])

(b) Printing process of MCC system (Courtesy [YYG+16])

Figure 2.8: Electron Beam Lithography and throughput enhancement

2.4.4 Multiple Electron Beam Lithography

Multiple Electron Beam Lithography (MEBL), as described in Section 1.3.4, is one of the popular Next-Generation Lithography (NGL) techniques that overcome the physical limits of 193 *nm* immersion lithography. It uses thousands to millions of electron beams (e-beams) in parallel to print the pattern on the wafer. Therefore, it overcomes the low throughput problem of Electron Beam Lithography (EBL) [Lin12] where only a single e-beam is used to print the patterns on the entire wafer.

To utilize multiple e-beams simultaneously, MEBL divides a layout into vertical stripes as shown in Figure 2.9(a), where routing segments present in separate stripes are printed by separate e-beams simultaneously and/or in different printing passes. The boundary of adjacent vertical stripes is called *stitch-lines* [LFC15]. During the layout printing process, due to the physical property of e-beams, adjacent e-beams deflect each other at the stitch-lines. These deflections cause overlay errors [LFC15] and generate pattern distortions near the stitch-lines. Figure 2.9(b) shows routing patterns that are cut by stitch-lines, and Figure 2.9(c) shows the pattern distortion caused by the overlay error after printing.

A *stitch-unfriendly region* τ [LFC15] is the region on either side of the stitch-line

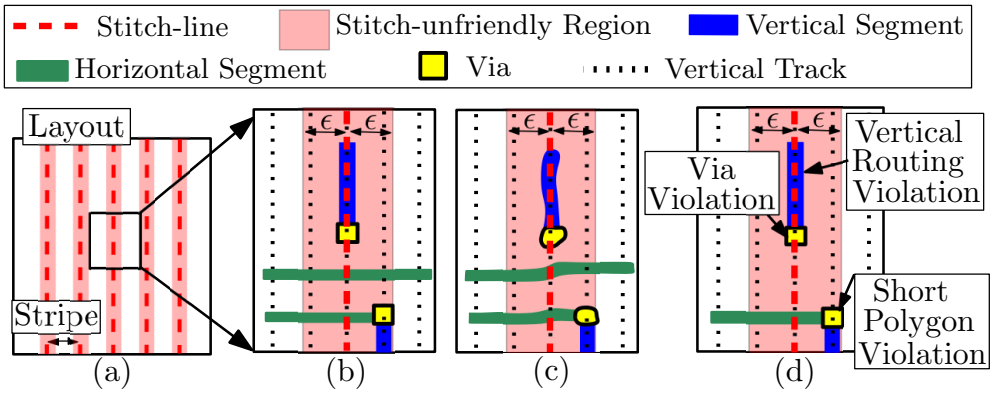


Figure 2.9: (a) Stitch-lines in a layout, (b) patterns cut by stitch-lines, (c) pattern distortion due to overlay error, and (d) violations due to a stitch-line. [SPBSK22]

up to ϵ horizontal distance from the stitch-line. Patterns that can suffer from overlay error due to adjacent e-beam deflection during the MEBL printing are referred to as *critical patterns*. These pattern distortions can lead to electrical violations including shorts, open circuits, and/or variations in line width, which in turn can degrade the timing, reliability, and also degrade performance of an IC. In more severe cases, these violations can lead to complete IC fabrication failure, resulting in costly re-manufacturing or yield loss.

To ensure high-quality fabrication outcomes, it is imperative that the VLSI physical design flow—especially the routing phase—takes extensive precautions to minimize the likelihood of pattern distortion arising from e-beam deflection. Routing strategies in MEBL must prioritize the careful handling of critical patterns, ensuring they are routed in a manner that avoids proximity to high-deflection zones near *stitch-lines*. This added constraint makes the routing phase significantly more complex in MEBL compared to traditional photolithography. Designers must not only consider standard routing metrics such as congestion, timing, and power, but also factor in lithographic reliability under e-beam exposure. As a result, routing algorithms must incorporate advanced models and heuristics to detect and mitigate potential e-beam-induced pattern distortions, adding an extra layer of challenge to the VLSI design process in MEBL environments.

Different types of routing patterns cut by a stitch-line produce varying levels of

pattern distortion [LFC15]. When horizontal segments are intersected by a stitch-line as shown in Figure 2.9(c), the resulting distortion is generally minimal. However, the following three types of violations can cause significant pattern distortions due to the overlay error:

- **Via violation**: It occurs when a via is on a stitch-line. During printing, the overlay error causes severe pattern distortion on the vias that are on the stitch-lines. Therefore, via violation is considered a hard constraint, and eliminating or reducing via violations is important to achieve better quality ICs.
- **Vertical routing violation**: It occurs when we place a vertical segment on a stitch-line. Like via violations, vertical routing violations also cause severe pattern distortions on the vertical segments that are on the stitch-lines. Therefore, it is also considered a hard constraint. Elimination or reduction in the number of vertical routing violations is also important to achieve better quality ICs.
- **Short Polygon violation**: It occurs when a horizontal wire segment cuts a stitch-line and lands on a stitch-unfriendly region with a leading via [LFC15]. Unlike via and vertical routing violations, the short polygon violation may not always lead to severe pattern distortion. Hence, it is considered as a soft constraint. However, violation reduction reduces the risk of pattern distortion.

Figure 2.9(d) shows these three violations. Therefore, to achieve a good quality IC, during routing in MEBL, these violations need to be minimized.

2.5 Existing Approaches to Mitigate Stitch-Line Violations

Several approaches have been proposed in the literature to mitigate stitch-line-induced violations, yet each comes with significant limitations.

The method in [LFC15] presented the first stitch-aware routing method for MEBL, which is a multilevel router that works in a bottom-up manner in two passes. It identifies three different types of stitch-line induced critical patterns that must be avoided in an MEBL-avoiding routing solution, and establishes corresponding stitch-aware constraints for different phases of routing. These constraints guide the routing

process to generate MEBL-compliant layouts.

In [LFC15] during global routing, the method distinguishes resource estimation differences between traditional optical lithography and MEBL, and integrates a global routing cost function with stitch-awareness into the router. The layer assignment is based on a segment conflict graph, achieving improved performance on an increment in the routing layers. For track assignment, they use two short-polygon-avoiding algorithms based on an ILP and on a graph with heuristic methods, where they discovered that the graph-based method offers higher efficiency and better suitability for iterative assignment. In the ILP-based approach, the track assignment aims to minimize the short polygon. They formulated it as a multicommodity flow problem and solved it using an ILP formulation. The graph-based approach first determines the order of net segments within each panel, followed by a resolution technique for bad ends with doglegs through a graph-based optimization procedure.

A stitch-aware detailed routing algorithm is further proposed in [LFC15], incorporating stitch-aware weighted grid costs and net ordering. It extends the conventional A*-search-based routing framework to perform routing between pin to segment or between multiple segments. Their experimental results demonstrate that [LFC15] reduces short polygons very significantly. However, it provides limited improvement in via violations. Overall, the stitch-aware routing framework effectively mitigates stitch-line-induced bad patterns, thereby enhancing manufacturability and facilitating MEBL technology development.

The perturbation-based post-routing method in [PBSK17] mitigates stitch-line violations by rerouting pre-routed nets. It reduces stitch-unfriendly patterns during the post-routing stage by perturbing wire segments, formulated as a maximum matching problem. [PBSK17] presents two variants where in *Variant-1*, vertical segments in a band are perturbed only if they cause one of the three stitch-line violations, while the rest remain fixed. In *Variant-2*, perturbation applies to stitch-unfriendly segments and to any regular segments located within their perturbation range.

As reported in [PBSK17], *Variant-1* achieves reductions of 17.41%, 16.46%, and 17.45% in routing, via, and short-polygon violations, respectively. In contrast, *Variant-2* achieves substantial improvements of 36.37%, 44.90%, and 91.65% in the corresponding violations and did not increase the total wirelength compared to the

initial layout. However, *Variant-2* incurs higher CPU time overhead. Moreover, since the method operates on completed layouts, it faces scalability challenges for large or congested circuits and tends to introduce additional vertical routing violations.

The dynamic programming-based placement method in [LYZ+17] presented a comprehensive study addressing stitch error minimization during the detailed placement phase. It optimizes the via positions and intra-cell vertical wires. Additionally, it accounts for local congestion, thereby providing the router with greater flexibility to reduce stitch errors in the upper metal layers. The method jointly minimizes stitch errors while optimizing traditional placement objectives such as density and wirelength. It employs a detailed placement engine that is swap-based, integrated with a stitch-aware placement approach for single-row.

The detailed placement algorithm in [LYZ+17] consists of two stages. In the first phase, a single-row-based approach optimally minimizes stitch errors and wirelength. If it eliminates all stitch errors, then the placement solution is finalized directly. Otherwise, during the second phase, cells are shifted and swapped to further improve wirelength and alleviate congestion. To solve the single-row detailed placement problem, the method adapts the dynamic programming-based algorithm proposed in [TLA+10]. A pruning technique with complexity $O(n \cdot M)$ is used in their method to accelerate the single-row optimization, where n denotes the number of cells in a row and M represents the maximum displacement. On average, this pruning strategy achieves approximately 30× speedup without any loss of optimality. Moreover, the pruning technique can be adjusted to further enhance local congestion optimization. Although the DP-based placement method in [LYZ+17] improves routability by refining local congestion, its effectiveness remains limited to violations arising from fixed pins and does not address stitch-related distortions in later routing stages.

The method in [HF20] proposed a router for a full chip that utilizes smart boundaries to minimize stitch-line violations. A stitch-aware track assignment algorithm was introduced to improve wire planning and enhance runtime efficiency. In addition, [HF20] developed a detailed routing technique with stitch-awareness by the insertion of stitch segments, serving as an important step to optimize smart boundaries and reduce patterns written with soft edges.

While prior studies primarily focus on developing different strategies for physical

design to prevent all three violations near stitch lines, method in [HF20] proposed novel stitching strategies such as soft edges and smart boundaries to address the stitching problem. These two techniques leverage the slight overlap between the ranges of adjacent e-beams. For instance, if each layout stripe has a width of 2.0 μm and each beam has a range of deflection extends to 2.2 μm with an overlapping region of 0.2 μm around the stitch-line, referred to as the *stitch-unfriendly region*.

[HF20] applied a smart boundary to patterns within the stitch region to allow a single e-beam to print the entire pattern. Conversely, if a segment passes through a stitch-unfriendly region, they apply a soft edge such that the segment can be printed by both beams with a gradual fade-out of exposure doses. Although the smart boundary method significantly enhances quality of routing patterns, it involves complex e-beam control.

The method in [HF20] evaluated their framework for routing by developing a conventional routability-driven router that did not consider smart boundaries. A comparison with prior stitch-aware routing methods was omitted because such techniques cannot operate with the smart boundary framework. The experimental results of [HF20] showed a reduction in the number of nets written with soft edges by over 97% and achieved comparable routability with acceptable wirelength and runtime costs. However, via counts increased by more than 2×, since a net crossing a stitch-unfriendly region requires at least two additional vias to realize a dogleg-like connection. Although this increase may degrade circuit performance, it improves manufacturability. When timing information is available, applying soft edges only to non-critical nets provides an effective trade-off between stitch control and performance. While the smart-boundary technique in [HF20] enhances stitch control through adaptive boundary definition, its reliance on complex e-beam adjustments limits scalability and practical deployment.

Overall, existing methods either operate too late in the flow, sacrifice one type of violation for another, or lack scalability to large designs. This highlights a critical research gap: the need for a lithography-aware routing framework that proactively minimizes multiple types of stitch-line violations simultaneously while maintaining scalability for modern, highly congested IC designs.

2.5.1 Comparative Analysis of Existing MEBL-Aware Methodologies

As discussed in Section 2.5, existing methodologies for mitigating stitch-line violations target different stages of the physical-design flow and consequently exhibit distinct trade-offs among manufacturability, routing quality, and computational complexity. To facilitate a systematic comparison, the representative MEBL-aware methodologies are analyzed in terms of their principal strengths and limitations. This analysis highlights the key trade-offs of existing approaches and identifies the research gaps that motivate the routing methodologies proposed in this thesis.

2.5.1.1 Stitch-Aware Detailed Placement Framework [LYZ+17]

Strengths: This methodology addresses stitch-related manufacturability concerns at the placement stage by relocating cells away from stitch-sensitive regions. By reducing the probability of stitch-induced defects before routing begins, it improves downstream routability and manufacturability.

Limitations: Because the optimization is performed before routing, routing-resource availability, congestion evolution, and detailed routing interactions are not explicitly considered. Consequently, routing-stage violations arising from net topology, layer assignment, congestion, or detailed-routing decisions cannot be directly addressed. The effectiveness of placement-based optimization therefore depends on how accurately placement-stage estimations correlate with actual routing behavior.

2.5.1.2 Stitch-Aware Routing Framework [LFC15]

Strengths: The methodology incorporates stitch awareness directly into the routing process by modifying routing costs near stitch-line regions. By discouraging routing through stitch-sensitive areas during route construction, it proactively reduces short-polygon violations and improves manufacturability compared to conventional routing approaches.

Limitations: The approach primarily focuses on stitch-aware path selection and does not explicitly optimize Steiner topology, stitch-aware routing capacities, or con-

gestion distribution. Consequently, routing flexibility is reduced in stitch-constrained regions, potentially leading to increased detours, higher congestion, and suboptimal wirelength. Furthermore, interactions between stitch avoidance and global-routing decisions are not comprehensively addressed.

2.5.1.3 Post-Routing Perturbation Framework [PBSK17]

Strengths: The framework operates after routing completion and therefore can be integrated into existing routing flows without requiring modifications to the routing engine. Through geometric perturbation of routing segments, it successfully removes a large number of stitch-line violations while largely preserving the original routing topology and wirelength characteristics.

Limitations: Since stitch violations are corrected after routing decisions have already been finalized, the framework is fundamentally reactive rather than preventive. The available solution space is therefore constrained by the existing routing topology. In highly congested designs, legal perturbations may be difficult to identify, and multiple iterative correction stages may be required, increasing computational overhead. Additionally, violations introduced by globally suboptimal routing decisions cannot be completely eliminated through local perturbation alone.

2.5.1.4 Smart-Boundary Routing Framework [HF20]

Strengths: The smart-boundary concept introduces flexible stitch-boundary manipulation to reduce pattern distortions and stitch-related defects. By dynamically adjusting boundary locations, the framework significantly improves manufacturability and reduces stitch-sensitive routing patterns.

Limitations: The additional flexibility provided by boundary manipulation introduces extra design complexity and often requires additional vias and routing resources. Consequently, improvements in stitch-related metrics may come at the expense of increased via count, larger routing overhead, and more complicated manufacturing control requirements. Furthermore, implementation requires enhanced e-beam control and boundary-management mechanisms that may increase process complexity.

2.5.2 Research Gap and Motivation

The above analysis reveals that existing MEBL-aware methodologies generally focus on a single stage of the design flow, such as placement, routing, or post-routing correction. Consequently, they often optimize stitch-related manufacturability at the expense of other objectives including wirelength, congestion, routing flexibility, via count, or runtime. This observation motivates the development of the routing methodologies proposed in Chapter 3 and 4, which integrate stitch awareness directly into global and detailed routing while simultaneously considering routability, congestion, wirelength, and manufacturability constraints.

This comparative analysis further clarifies the strengths and limitations of existing methodologies and highlights the primary research gap addressed by this thesis: the development of scalable routing techniques that simultaneously optimize routing quality, manufacturability and computational efficiency, while remaining compatible with emerging lithographic technologies and advanced physical-design flows.

To evaluate the techniques proposed in this thesis, a baseline routing framework is employed. The following section presents the baseline routing framework used in Chapter 3 and 4.

2.6 Baseline Routing Framework

To address the research gap identified in the previous section, this thesis develops stitch-avoiding global and detailed routing methodologies. Our proposed stitch-avoiding global router, described in Chapter 3, is inspired by the traditional global router *BoxRouter* [CP06], while the stitch-avoiding detailed router, described in Chapter 4, is based on *RegularRoute* [ZC12]. Both *BoxRouter* and *RegularRoute* were originally developed for conventional photolithography and are thus unsuitable for next-generation lithography techniques such as MEBL. The following sections briefly describe these methods, which also serve as baseline frameworks for comparison with our proposed stitch-avoiding routing approach.

2.6.1 BoxRouter: A Traditional Global Router

BoxRouter [CP06] is a global router that employs congestion-driven expansion of the box, a maze router with adaptive capabilities, and integer linear programming that progressively routes nets within a box. It begins with a *Pre-Routing* step to accurately identify the congested regions. From this hotspot, BoxRouter [CP06] progressively expands a box that eventually covers the entire circuit. Routing within each box iteration is performed using an ILP and maze-routing-based methods. A progressive ILP-based method called PILP achieves higher efficiency than conventional ILP in both speed and quality, while the adaptive quality-based maze routing (AMR) on a multi-source with multiple-target model—effectively reduces congestion and wirelength. Finally, a *Post-Routing* is applied to reroute except rip-up to balance wirelength and routability.

BoxRouter divides the circuit into two parts after each expansion: inside and outside the box. As illustrated in Figure 2.10(a), the wires labeled (a, b) inside the box are assigned greater routing priority compared to the external ones (c, d), since detours for internal wires increase wirelength and external wires can often find alternate routes. Hence, BoxRouter prioritizes internal routing using PILP to maximize box capacity, and uses AMR for unrouted wires with following strategies:

1. **Inside the box:** greedily utilize routing capacities since inner wires have higher priority.
2. **Outside the box:** use routing capacities with restraint for future viable paths.

These strategies maintain balanced wire density and minimize wirelength overhead,

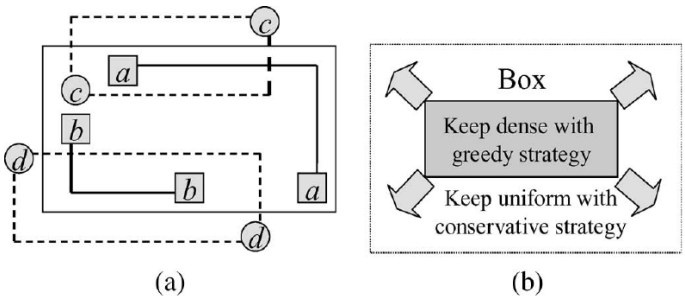


Figure 2.10: Box Routing concept, (a) Working principle of BoxRouting, (b) strategies of Routing. (Courtesy [CP06])

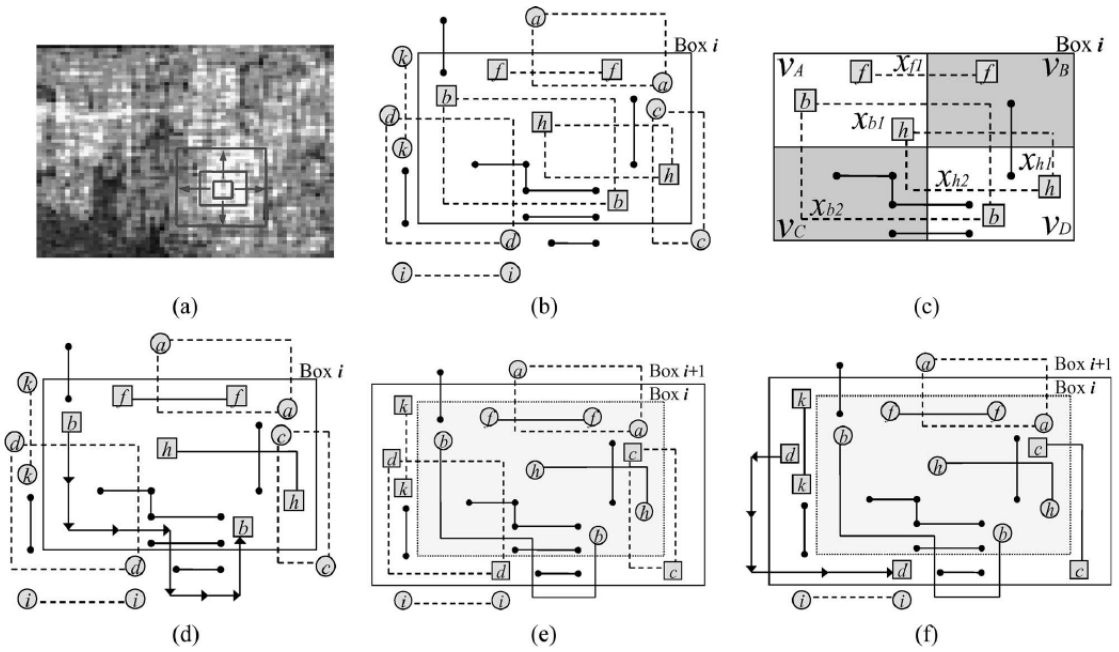


Figure 2.11: BoxRouting flow. (a) Initial hotspot box. (b) Wires in box i. (c) PILP routing. (d) AMR for unrouted wires. (e–f) Box expansion and rerouting. (Courtesy [CP06])

as illustrated in Figure 2.10(b).

BoxRouting process: As shown in Figure 2.11, BoxRouting iteratively performs three steps—Progressive ILP routing, Maze Router with adaptive features, and expansion of box-till it covered the entire circuit.

Consider a box, it expands from the region marked as the most congested region as shown in Figure 2.11(a), consider Figure 2.11(b), where wires in the box after the ith expansion are represented by squares (b, f, h), and the remaining wires are shown with circles. Already-routed wires from either *Pre-Routing* or previous BoxRouting steps are depicted as solid lines. Some flat wires (e.g., f, i, k) may remain unrouted if *Pre-Routing* failed due to overflow or if new Steiner points introduced by AMR converted a non-flat wire into a flat one.

To keep ILP size within limits, first, wires within the current box are processed by Progressive ILP and AMR. In Figure 2.11(c), these wires are represented by G-cells

$$\max \sum \{x_{i1} + x_{i2}\} \quad \forall i \in W_{box} \tag{2.11}$$

subject to:

$$x_{i1} + x_{i2} \leq 1 \qquad \forall i \in W_{box} \tag{2.12}$$

$$x_{i2} = 0 \qquad \forall i \in W_{box} \cap W_{flat} \tag{2.13}$$

$$\sum_{e \in x_{ij}} x_{ij} \leq c_e \qquad \forall e \in W_{box} \tag{2.14}$$

$$x_{i1}, x_{i2} \in \{0, 1\} \qquad \forall i \in W_{box} \tag{2.15}$$

Figure 2.12: General PILP formulation. (Courtesy [CP06])

(v_A, v_B, v_C, v_D) and solved using the PILP to maximize routability. To minimize via count, the method considered for each wire, two L-shaped candidates for routing denoted by x_{b1}, x_{b2} and x_{h1}, x_{h2}, while flat wires have only one candidate (x_{f1} with $x_{f2} = 0$).

PILP formulation: For each expanded box, only unrouted wires within it are considered. Each wire has two L-shaped routing candidates, while flat wires have only one. The PILP formulation for the general case is given in Figure 2.12 with c_e denoting capacity on edge e for routing, the set of unrouted wires and flat wires within the box are denoted by W_{box} and W_{flat}, respectively.

Performance: Experimental results demonstrate that BoxRouter significantly outperforms existing global routers—achieving 91% better routability than Labyrinth. This method also obtained 14% less wirelength with 3.3× speedup. Compared with the Chi-dispersion router, it obtains a routability of 79%, similar wirelength and 2× speedup. This method also has a 16× speedup and 4.2% less wirelength compared to a multicommodity-flow-based router with comparable routability.

2.6.2 RegularRoute: A Traditional Detailed Router

One of the most efficient traditional detailed routers, namely *RegularRoute* [ZC12], performs routing with regular patterns and ensures greater design rule compliance using a methodology called correct-by-construction. Given design rules that are based

on spacing, a solution of global routing, and routing track information for 3-D layers, it produces a detailed routing solution for each successive layers starting from the bottom layer. Vertical spine routing topology (VSRT) is used to route local nets confined within a single G-cell. The routing usage in this phase is treated as a blockage during global segment assignment.

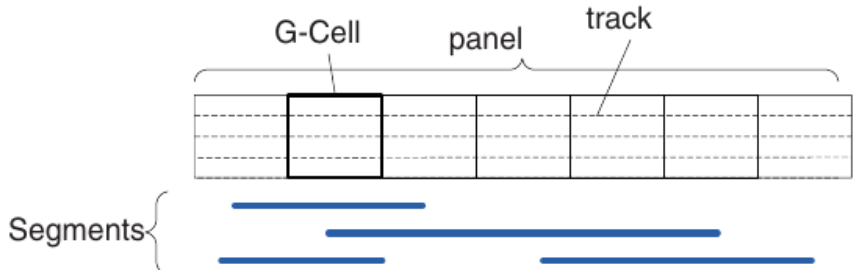


Figure 2.13: Track, segment, and panel definitions (Courtesy [ZC12])

In each layer, [ZC12] formulates the global segment assignment within a panel as shown in Figure 2.13 as a Maximum Weighted Independent Set (MWIS) problem and solves it with an efficient heuristic. Using a greedy approach, the method performs partial routing for the unassigned segments, and the remaining segments are deferred to upper layers by promoting their terminals. In the top-most layer, merging of panels and maze routing enhance routability. Optimization phases are applied locally on the basis of local shifting and also rerouting refines the solution to satisfy all design rules.

The MWIS problem is NP-complete [GJ+90], therefore, RegularRoute employs a fast heuristic exploiting the conflict graph structure. The benefit of assigning a vertex v is defined as:

$$B(v) = W(v) - \beta W_i(v) - \gamma W_o(v), \tag{2.16}$$

where $W(v)$ denotes vertex weight, $W_i(v)$ is the sum of weights for same-segment conflicts, $W_o(v)$ for different-segment conflicts, β and γ balance their influence:

$$\beta = 0.4 \times \frac{W(v)}{\max(W_i(v), W_o(v))}, \quad \gamma = 0.2 \times \frac{W(v)}{\max(W_i(v), W_o(v))} \tag{2.17}$$

Vertices are ranked by decreasing $B(v)$; the vertex with the highest benefit is

assigned first, followed by conflict graph updates. This prioritizes segments with high weight (criticality), fewer conflicts between same segments (less flexibility), and fewer conflicts between different segments (lower congestion risk).

Due to the lack of standard detailed routing based on grid, RegularRoute introduced two benchmark sets derived from the ISPD98 and ISPD05/06 placement suites. The experimental results demonstrated its high efficiency and effectiveness in producing regular, design-rule-compliant detailed routing solutions.

The routing frameworks discussed in this section represent traditional algorithmic approaches that have long formed the foundation of physical design automation. As VLSI designs continue to grow in complexity, machine learning has emerged as a complementary paradigm for learning optimization strategies directly from design data. The following section reviews these learning-based techniques, which underpin the methodologies presented in the later chapters of this thesis.

2.7 Machine Learning for Physical Design Automation

Machine learning (ML) has increasingly been applied to various phases of physical design, lithography hotspot detection, 3D IC design, and others. These approaches aim to accelerate design processes, improve the quality of results, and enable predictive modeling for complex metrics such as congestion, wirelength, and signal integrity. In this section, we review several notable contributions.

Recent advances in artificial intelligence have significantly transformed VLSI electronic design automation. Machine learning techniques are increasingly being applied to various physical-design stages, including placement, routing, congestion prediction, timing optimization, and lithography hotspot detection. Li *et al.* [LCZ22] presented a comprehensive survey of machine-learning-based routing methodologies, categorizing existing approaches into congestion prediction, routability estimation, routing optimization, reinforcement learning, and graph-learning-based routing. The survey highlights the growing transition from traditional heuristic-driven routing algorithms toward intelligent learning-assisted optimization frameworks.

2.7.1 ML for Floorplanning and Placement Heuristics

Zhuolun et al. [HMZ+20] formulated floorplanning as a reinforcement learning problem, learning effective local search heuristics inspired by simulated annealing. The RL agent selects moves in a large solution space, guided by rewards based on energy reduction and adversarial penalties for inefficient moves. Deep Q-learning is used to train the agent, enabling automated and high-quality floorplan generation across benchmarks of varying sizes.

2.7.2 ML for Global Placement and Routability Optimization

Liu et al. [LSL+21] proposed a deep learning-based model to improve routability in global placement. They developed a Fully Convolutional Network (FCN) that is used to predict congestion heatmaps for routing from placement solutions. The predicted congestion map is incorporated into the placement objective function, enhancing routing-aware optimization in DREAMPlace [LDL+19]. The model uses three input feature maps: Rectangular Uniform Wire Density (RUDY), pin density (PinRUDY), and macro region coverage. Experiments on ISPD 2015 benchmarks demonstrated a reduction in total congestion rate up to 9.05% and in routed wirelength of about 5.3% as compared to DREAMPlace, showing the effectiveness of their approach in predicting congestion patterns.

Mirhoseini et al. [MGY+21] introduced a graph-based methodology for fast macro placement using Deep Reinforcement Learning (DRL). Starting from an unplaced netlist, an RL agent sequentially places macros based on a policy network trained with a reward function considering wirelength, congestion, and density. Standard cells are clustered using hMETIS and placed via a force-directed method. The approach employs an Edge-GNN model to embed netlist information and a neural network to predict rewards, enabling generalization to unseen designs. This methodology was applied successfully to Google TPU and RISC-V CPU designs, outperforming traditional simulated annealing-based methods in both wirelength and congestion.

Pan et al. [LDL+19] presented DREAMPlace, a GPU-accelerated analytical placement framework. The method draws an analogy between neural network training and electrostatics-based placement, modeling netlist nodes as charged particles. The

framework leverages deep learning toolkits to accelerate placement computations and efficiently calculate wirelength and density costs, demonstrating high performance on both benchmark and industrial designs [LDL+19, LCC+15, CKKW18].

Xie et al. gave a convolutional neural network (CNN)-based routability prediction method called RouteNet [XHF+18] for mixed-size designs. It solves two problems:

(a) for cell placement, it generates a fast routability forecast using Design Rule Violations (DRV) counts,

(b) prediction of design rule checking (DRC) hotspot location to proactively modify a few identified solutions and prevent design rule violations.

2.7.3 ML for Routing, Routing Congestion and Routability Prediction

Recent advances in VLSI routing have increasingly integrated machine learning techniques to complement conventional heuristic-driven routing algorithms. Unlike traditional approaches that rely primarily on graph search, maze routing, Steiner-tree heuristics, integer linear programming, and negotiated congestion, learning-based methods exploit design data to predict routing behaviour, guide optimization decisions, and improve routing quality. These approaches have demonstrated significant improvements in wirelength, congestion, routability, timing closure, and design-rule compliance.

Machine learning has been applied across multiple stages of the routing flow. During global routing, neural-network-guided frameworks learn congestion-aware routing patterns to reduce wirelength and routing hotspots [LZD+24, CSD+24]. Reinforcement learning has also been employed for sequential Steiner-tree construction, where routing decisions are learned by balancing objectives such as routing capacity, delay, and wirelength [LLH+24]. Hybrid methodologies further integrate learning models with conventional optimization techniques, such as maze routing and integer linear programming, enabling learned predictions to guide routing decisions while preserving the robustness of traditional optimization algorithms [LCZ22, PXZC12].

Cheng *et al.* [CLL+22] proposed a unified framework that combines policy-gradient reinforcement learning for placement with generative routing neural networks

for routing optimization. Unlike conventional approaches that treat placement and routing as separate optimization stages, their framework jointly learns placement and routing strategies, demonstrating the increasing integration of deep learning across multiple stages of the physical-design flow. Their work further highlights the potential of generative neural models for solving complex routing optimization problems.

Chen *et al.* [CKZ+23] proposed PROS 2.0, a deep-learning-based plug-in for routability optimization and routed wirelength estimation. By leveraging learned routing knowledge during placement, the framework improves routability and wirelength estimation with negligible runtime overhead, demonstrating the practical integration of machine learning into industrial EDA workflows.

Machine learning has also been investigated for intermediate routing stages. During track assignment, learning-based approaches predict suitable routing-track allocation patterns from congestion information, thereby improving signal integrity and routing-resource utilization [LCZ22]. In detailed routing, learning-based methods assist in improving routing quality while satisfying spacing, layer-assignment, and design-rule constraints [LZD+24].

Overall, recent routing research demonstrates a clear transition from purely heuristic optimization towards intelligent learning-assisted routing frameworks. These methodologies combine the scalability of traditional routing algorithms with the predictive capability of machine learning models, providing improved routability, congestion management, and timing performance for large-scale VLSI designs. This growing adoption of machine learning in routing provides the motivation for the routing methodologies proposed in this thesis.

Yu et al. [YZ19] proposed a cGAN framework to forecast detailed routing congestion from post-placement data. Unlike hotspot detection, it predicts full congestion heat maps, enabling early-stage congestion assessment. The model benefits from transfer learning to adapt to new designs, improving robustness and accuracy. Alawieh et al. [ALL+20] extended this concept for large-scale FPGAs, demonstrating that prior cGAN-based methods were limited to small net counts. It scales to hundreds of thousands of nets, addressing image sparsity issues encountered in earlier models.

More recently, Li *et al.* [LHW+24] proposed an Inception-boosted U-Net architecture for routability prediction and design-rule-checking hotspot prediction. By

combining multi-scale feature extraction with encoder-decoder learning, the proposed framework accurately predicts routing bottlenecks and DRC hotspots before detailed routing, thereby reducing routing iterations and improving design quality.

2.7.4 ML for Crosstalk and Signal Integrity Prediction

Crosstalk-induced noise and delay are critical concerns in modern ICs. Liang et al. [LXJ+20] proposed routing-free crosstalk prediction models that identify problematic nets at the placement stage, avoiding expensive trial routing. Features include RUDY-based wire density, net-topology metrics (e.g., HPWL, fan-out, max source-sink distance), electrical and timing information. Multiple ML models—logistic regression, neural networks, random forests, and XGBoost—were trained for three classification tasks: high coupling capacitance, high crosstalk noise, and significant incremental delay. Graph-based ML models using dual circuit and proximity graphs further improved prediction performance. Experiments on IWLS 2005 benchmarks showed that XGBoost achieved the best results across tasks.

2.7.5 ML Methods in Lithography Hotspot Detection

Machine learning (ML) techniques have been widely applied in lithography hotspot detection to increase prediction accuracy and decrease false alarms. These methods primarily focus on designing deep learning architectures capable of learning discriminative layout features from hotspot and non-hotspot instances.

Jiang et al. [JYY+20] proposed Binarized Neural Networks (BNNs) for hotspot detection, which is a deep learning architecture-based model. They designed a Binarized Residual Neural Network to improve detection accuracy while maintaining low computational cost. To further enhance performance, the authors employed ensemble learning, combining multiple models to achieve better hotspot detection than individual networks. Additionally, they adopted a biased learning technique to address the challenge of imbalanced datasets, which contain significantly more non-hotspot instances than hotspots. Experiments on the ICCAD 2012 benchmark demonstrated that the ensemble model improved detection accuracy, although it slightly increased false alarms compared to single-model approaches.

Geng et al. [GYZ+20] proposed an attention mechanism-based deep convolutional neural network to learn feature embeddings of layout patterns. The model is designed to capture optimal layout feature representations that can effectively distinguish hotspots from non-hotspots, while classifying hotspot patterns with high accuracy and minimal false alarms, and maintaining low runtime overhead. The attention module sequentially infers an attention map for a 1D channel, followed by a two-dimensional spatial attention map, which is then multiplied element-wise with the input feature maps. This mechanism enables the network to focus on salient layout regions, improving hotspot detection performance.

Ye et al. proposed LithoGAN [YALP19], a lithography modeling framework for end-to-end operation based on generative adversarial networks (GANs). The framework accurately predicts resist patterns from given mask patterns, achieving a significant decrease in time compared to the conventional lithography simulation and other machine learning approaches. It decomposes the modeling task into resist shape modeling and resist center prediction. Input mask and resist pattern pairs are processed as images, cropped, and converted to an appropriate resolution for the model. The framework is evaluated using metrics such as Edge Displacement Error (EDE), Pixel Accuracy, Class Accuracy, and Mean Intersection over Union (IoU). Experiments show that LithoGAN provides superior accuracy and runtime performance for N10 and N7 technology nodes. Other recent works in hotspot detection include:

- Yang et al. [YZM+21] proposed mask optimization methods using deep learning.
- Chen et al. [CZY+19] developed a faster region-based hotspot detector.
- Yang et al. [YLM+18] introduced GAN-based OPC for mask optimization.
- Jiang et al. [JLM+20, JLM+21] proposed Neural-ILT for mask printability and complexity co-optimization.
- Yang et al. [YPG+19] addressed multi-layer hotspot detection using adaptive squish patterns.

Overall, ML techniques in EDA span supervised, reinforcement, and generative approaches. They have demonstrated significant improvements in placement quality, congestion prediction, and signal integrity analysis. While these methods offer promising results, challenges remain in scaling to extremely large designs, handling diverse design constraints, and integrating seamlessly into industrial EDA flows.

2.7.6 Categories of Machine Learning Methods for Physical Design Problems

The literature on machine-learning-based physical design encompasses a diverse range of learning paradigms that have been applied to various optimization tasks, particularly routing. For systematic comparison, the reviewed studies are categorized according to the underlying learning paradigm rather than the specific application. The principal strengths and limitations of each category are discussed below.

2.7.6.1 CNN-Based Congestion Prediction Methods [YZ19, ALL+20, LSL+21]

Strengths: Convolutional neural networks effectively exploit the spatial locality present in placement and routing layouts. By learning hierarchical spatial features from placement density maps, pin distributions, and routing-resource utilization patterns, CNN-based models can accurately predict routing congestion and routability hotspots before routing is performed. These predictions enable congestion-aware placement and reduce routing iterations.

Limitations: CNN-based models primarily operate on image-like representations and capture spatial correlations more effectively than connectivity information. Since netlist topology and long-range interconnect dependencies are not explicitly represented, their ability to model complex interactions among distant circuit components is limited. Consequently, prediction accuracy may degrade when congestion is strongly influenced by global netlist structure rather than local geometric features.

2.7.6.2 Reinforcement-Learning-Based Methods [MGY+21, LCY21, LLH+24]

Strengths: Reinforcement learning formulates physical-design optimization as a sequential decision-making problem, enabling direct optimization of design objectives such as wirelength, congestion, timing, and routability. Unlike supervised-learning approaches, RL agents learn optimization strategies through interaction with the design environment and can discover non-trivial design policies that are difficult to encode using handcrafted heuristics. Recent results demonstrate that RL-based approaches can achieve near-optimal placement and Steiner-tree construction quality.

Limitations: The training process requires a large number of environment interactions and substantial computational resources. Performance is highly dependent on reward-function design, exploration strategy, and training stability. Furthermore, sparse rewards, delayed objective feedback, and large combinatorial action spaces often result in long training times and challenging convergence behaviour. Generalization to unseen designs may require retraining or fine-tuning.

2.7.6.3 Graph-Neural-Network-Based Approaches [LPZ+20]

Strengths: Graph neural networks naturally operate on graph-structured representations such as netlists, routing-resource graphs, and Hanan-grid graphs. Through iterative message passing, GNNs can explicitly model connectivity relationships, structural dependencies, and information propagation across neighboring entities. This capability makes them particularly suitable for routing, partitioning, congestion prediction, and Steiner-tree generation tasks where connectivity plays a central role. Compared with CNN-based approaches, GNNs provide a richer representation of circuit topology and interconnect structure.

Limitations: The computational cost of message passing increases with graph size and edge density, leading to higher memory consumption and training complexity for large-scale circuits. Deep GNN architectures may also suffer from over-smoothing, over-squashing, and long-range information propagation challenges. In addition, learned graph representations often remain difficult to interpret, limiting explainability and debugging capabilities.

2.7.6.4 Hybrid Machine-Learning-Assisted Optimization Frameworks [MGY+21, LLH+24]

Strengths: Hybrid frameworks integrate predictive learning models with optimization engines such as analytical placement, maze routing, Steiner-tree construction, or mathematical programming techniques. Such approaches leverage machine learning for prediction, guidance, or decision prioritization while preserving the robustness and constraint-handling capabilities of traditional optimization algorithms. Consequently, they often achieve superior scalability, improved solution quality, and stronger gen-

eralization compared with purely heuristic or purely learning-based approaches.

Limitations: The effectiveness of hybrid frameworks depends heavily on the quality of interaction between the learning component and the underlying optimization engine. Errors in learned predictions may propagate through subsequent optimization stages, potentially degrading final solution quality. Integrating machine learning models into existing EDA flows increases implementation complexity and may require careful calibration to ensure consistent performance across diverse design instances.

The following section reviews the existing approaches for Rectilinear Steiner Minimum Tree (RSMT) construction, which forms the foundation of the stitch-avoiding global and detailed routing frameworks presented in Chapter 3 and 4. As the solution quality of these routing frameworks largely depends on the quality of the underlying RSMT, a review of existing construction techniques is essential.

2.8 Existing Approaches for RSMT Construction

As discussed in Section 1.4, a *Rectilinear Steiner Minimum Tree* (RSMT) connects a set of d points (the pins of a net) in the 2D plane with the minimum possible total length, potentially introducing extra Steiner points to shorten the routing path. The *degree* is defined as the pin count of a net. An RSMT minimizes the total wirelength measured in the L_1 (Manhattan) distance metric and serves as a backbone for many popular global routing algorithms.

In [Han66], Hanan proved that an optimal RSMT always exists on the *Hanan grid*, which is formed by drawing horizontal and vertical lines through each of the d pins of a net, as illustrated in Figure 5.1. The Hanan grid contains d^2 grid points. The solver `GeoSteiner` [JWWZ18] can generate optimal RSMTs, but its runtime grows exponentially with the net degree.

In contrast, `FLUTE` [CW07] is a widely used heuristic that constructs RSMTs very efficiently and is incorporated into many state-of-the-art global routers [CLYP09, SPBSK22]. FLUTE employs a pre-computed lookup table to generate optimal solutions for nets up to degree 9, and applies a net-breaking heuristic for larger nets, which may lead to suboptimal results. The first sub-exponential algorithm for opti-

mal RSMT construction was introduced in [FLK+20], using dynamic programming together with the bounded treewidth property.

More recently, machine learning–based approaches for RSMT construction have attracted attention. `REST` [LCY21] applies reinforcement learning to construct RSMTs, achieving on average $\leq 0.36\%$ error in wirelength for nets up to degree 50. `REST` employs a progressive training scheme in which the model for degree-n nets depends on a pre-trained model for degree-$(n-1)$ nets. This dependency causes training time to grow rapidly with net degree; for example, the increase in training time from degree 50 to 51 is about $10\times$ larger than the increase from degree 10 to 11.

We can find a square grid called *Hanan grid* by joining vertical and horizontal line segments passing through each pin of a net. According to [Han66], an optimal RSMT only contains a particular subset of edges of the Hanan grid. Figure 5.1 shows a Hanan grid of a net and its optimal RSMT formed only by the edges of the Hanan grid. The primary challenge to find an optimal RSMT for a net is that it is NP-hard [Har82]. Hence, there are existing techniques that either generate optimal RSMTs in exponential/sub-exponential time or near-optimal RSMTs in polynomial time using either heuristics or machine learning methods. Current state-of-the-art methods for RSMT construction can be classified into the following four categories:

- Methods like `GeoSteiner` [JWWZ18] and [FLK+20] generate optimal RSMTs in exponential and sub-exponential time, respectively. The method in [FLK+20] is based on dynamic programming (DP) and the bounded treewidth property.
- `FLUTE` [CW07] is one of the most commercially used heuristic methods for generating non-optimal RSMTs in polynomial time. For shorter nets (degree ≤ 9), a pre-computed lookup table is used to generate RSMTs, and for larger nets (degree > 9), a net-splitting heuristic is used.
- `REST` [LCY21] is the most popular machine learning method to generate near-optimal RSMTs in polynomial time. `REST` [LCY21] uses reinforcement learning to construct RSMTs, and it outperforms the heuristic method `FLUTE` [CW07] in terms of wirelength error and generation time. Lin *et al.* [LLH+24] proposed a unified reinforcement-learning framework that simultaneously considers routing objectives and design constraints. Apart from these two methods, methods like [ATS+21] and [DYXZ21] solve the single Steiner tree construction problem

for Euclidean distance using Deep Reinforcement Learning and Graph Neural Networks .

- For a given set of pins on the Hanan grid, the method in [LK19] presents an algorithm to construct all possible RSMTs by building a database called *ARSMT DB*. In other words, it generates all possible RSMTs of a single net.

Table 2.1: Comparison of Existing RSMT Generation Methods

Method	**Strengths**	**Limitations**
GeoSteiner [JWWZ18]	Produces optimal RSMTs and remains the benchmark for solution quality.	Suffers from exponential growth in the number of candidate Full Steiner Trees with increasing net degree, resulting in substantial runtime and memory overhead for large routing instances.
Subexponential Dynamic Programming Approaches [FLK+20]	Provide theoretically improved exact algorithms through bounded-treewidth formulations while maintaining optimality guarantees.	Despite theoretical improvements, computational requirements remain significant for large routing instances and higher-degree nets.
FLUTE [Chu04]	Offers extremely fast runtime and has become one of the most widely adopted Steiner-tree generation algorithms in industrial physical-design flows.	Uses decomposition and lookup-table approximations, which may introduce wirelength suboptimality for larger net degrees.
REST [LCY21]	Shows that reinforcement learning can generate near-optimal RSMTs with significantly lower wirelength error than conventional heuristics.	Requires substantial training effort and employs progressive model generation, causing training complexity to increase with net degree.

2.8.1 Comparative Analysis of Existing RSMT Construction Methodologies

Table 2.1 summarizes the strengths and limitations of representative RSMT construction methods, highlighting the motivation for the learning-based approaches proposed in this thesis.

2.8.2 Research Gap and Motivation

The above analysis indicates that existing RSMT construction methods continue to face a trade-off between solution quality, computational efficiency, and scalability. Exact algorithms provide optimal solutions but incur high computational cost, whereas heuristic methods achieve fast runtime at the expense of solution quality for larger nets. Although recent machine-learning-based approaches have demonstrated the potential of learning-based optimization, they are primarily limited to single-net RSMT construction and often rely on computationally intensive training strategies.

These limitations motivate the research presented in Chapter 5 and Chapter 6 of this thesis. Chapter 5 proposes a conditional generative neural network-based method for efficient single-net RSMT construction, while Chapter 6 presents a graph-neural network based concurrent generation of RSMTs for multiple nets, enabling the model to capture inter-net relationships and improve scalability for practical VLSI routing applications.

2.9 Summary

This chapter reviewed the background and related work relevant to the research presented in this thesis. It introduced the physical-design flow, conventional and next-generation lithography techniques, existing MEBL-aware routing methodologies, traditional routing frameworks, machine learning applications in physical design, and existing approaches for RSMT construction. The literature review identified the limitations of existing methods discussed in Section 2.5 and 2.8, thereby motivating the development of the proposed stitch-avoiding routing methodologies presented in

Chapter 3 and 4, as well as the learning-based RSMT construction techniques presented in Chapter 5 and 6. Building on this background, the next chapter presents the proposed stitch-avoiding global routing framework for Multiple Electron Beam Lithography (MEBL).

CHAPTER

THREE

STITCH-AVOIDING GLOBAL ROUTING FOR MULTIPLE E-BEAM LITHOGRAPHY

Contents

3.1 Introduction **65**

3.2 Motivation **66**

3.3 Preliminaries and Problem Formulation **66**

3.3.1 Preliminaries 66

3.3.2 Problem Formulation 70

3.4 Overview of Our Proposed Global Router for MEBL **71**

3.5 Phase-I: Stitch-avoiding Steiner Minimum Tree (SSMT) **71**

3.6 Phase-II: Multistage Stitch-avoiding Global Routing **75**

3.6.1 Stitch-avoiding Global Routing of Two-Pin Flat Nets 76

3.6.2 Stitch-avoiding Global Routing for Multi-Pin Nets and TPBN . . . 76

3.6.3 Stitch-constrained Maze Routing 89

3.7 Evaluation of our Stitch-avoiding Global Router **94**

3.7.1 Discussion on the Optimality and Scalability of the Proposed Routing Framework . . . 98
3.7.2 Limitations and Correctness of the Proposed Routing Framework . 100
3.8 Concluding Remarks . . . 101

3.1 Introduction

Multiple Electron Beam Lithography (MEBL) is a prospective NGL technique that addresses the limitations of 193 *nm* immersion lithography and the throughput limitation of Electron Beam Lithography (EBL). As described in Section 2.4.4, MEBL uses thousands and potentially millions of parallel e-beams to draw patterns on wafers efficiently. To utilize multiple beams simultaneously, layouts are divided into vertical stripes as shown in Figure 2.9(a), where each e-beam prints one stripe. The boundaries between stripes, called *stitch-lines* [LFC15], are prone to overlay errors due to mutual e-beam deflection, leading to pattern distortions near *stitch-lines* as shown in Figure 2.9(b)–(c).

Different routing patterns exhibit different levels of stitch-line distortions [LFC15]. As shown in Figure 2.9(d), the most critical are: (1) *vertical routing violations*, where vertical segments overlap stitch-lines, (2) *via violations*, where vias lie directly on stitch-lines—both causing severe distortions and being treated as hard constraints, (3) *short polygon violations*, where horizontal segments intersect stitch-unfriendly regions with a leading via, considered a soft constraint. To ensure high-quality fabrication, the routing phase in MEBL must minimize these violations while balancing traditional metrics such as congestion, timing, and power. This makes MEBL-aware routing significantly more complex, demanding advanced modeling and heuristics to mitigate e-beam-induced pattern distortions during physical design.

In this chapter, we consider mitigating the pattern distortions caused by these three violations in the global routing phase. Therefore, we present a stitch-avoiding global routing framework for MEBL that explicitly models stitch-line avoidance during global routing, enabling traditional detailed routers to generate solutions with

minimal stitch-line violations. Evaluations on the Faraday [oCADI21, ACR+04] benchmark suite indicate a notable reduction in stitch-line violations.

3.2 Motivation

In the post-routing phase, the method in [PBSK17] mitigates stitch-line violations by performing wire perturbation and rerouting of pre-routed nets. Because it operates on a pre-routed layout with only local perturbations, it may be unable to avoid stitch-line violations in congested and/or large circuits. The method in [LFC15] employed a traditional global router augmented with a new congestion constraint, achieving most of its violation reduction during the detailed routing phase. Therefore, a dedicated global router for MEBL that considers the existence of the stitch-lines during global routing is not available in the literature. Given the strong dependency of detailed routing on solution quality of global routing, developing an MEBL-specific global router is crucial for producing detailed routing solutions free of stitch-line violations.

3.3 Preliminaries and Problem Formulation

This section presents the necessary preliminaries, formally defines the problem under consideration, and provides an overview of the proposed methodology to address it.

3.3.1 Preliminaries

We introduce the formal definitions of essential concepts and net classification required to model and analyze the routing problem in MEBL. Consider a netlist $\mathcal{N} = \{N_1, N_2, \ldots, N_n\}$ for routing. Nets are grouped according to their pin count into two-pin nets and multi-pin nets, with definitions provided in Definition 3.3.7 and Definition 3.3.8. For two-pin nets, pin placement further distinguishes them into flat nets and bend nets, as defined in Definition 3.3.9 and Definition 3.3.10.

Definition 3.3.1 (Grid graph model [She99])**.** In this model, a global routing instance is captured by a graph $G = (V, E)$. The vertex set V corresponds to the global grid cells (G-cells), with each vertex $v_r \in V$ representing a G-cell c_r. The edge set $E = e_{rs}$

contains an edge between v_r and v_s whenever the corresponding G-cells c_r and c_s are adjacent. Every edge e_{rs} is assigned a routing capacity C_{rs}, which denotes the maximum number of net segments that may traverse the boundary between G-cells c_r and c_s.

Definition 3.3.2 (Stitch-holding cell)**.** A G-cell is considered a *stitch-holding cell* if it contains a stitch-line.

Definition 3.3.3 (Stitch-avoiding routing capacity)**.** The *stitch-avoiding routing capacity* C_{rs}^{Stitch} of an edge e_{rs} is defined as

$$C_{rs}^{\text{Stitch}} = (C_{rs} - |L|)$$

when e_{rs} is a vertical edge and both c_r and c_s are stitch-holding cells, where $|L|$ denotes the number of vertical metal layers available for detailed routing. Otherwise,

$$C_{rs}^{\text{Stitch}} = C_{rs}.$$

Definition 3.3.4 (stitch-avoiding available routing capacity)**.** The *stitch-avoiding available routing capacity* A_{rs}^{Stitch} for an edge e_{rs} is defined as

$$A_{rs}^{\text{Stitch}} = C_{rs}^{\text{Stitch}} - U_{rs},$$

where U_{rs} denotes the number of net segments that cross the G-cell boundary corresponding to edge e_{rs}.

Definition 3.3.5 (Stitch-avoiding vertex capacity)**.** The *stitch-avoiding vertex capacity* C_r^{Stitch} represents the maximum number of pins or vias that may be placed in the G-cell c_r.

Definition 3.3.6 (Stitch-avoiding available vertex capacity)**.** The *stitch-avoiding available vertex capacity* A_r^{Stitch} for a vertex v_r is defined as

$$A_r^{\text{Stitch}} = C_r^{\text{Stitch}} - U_r,$$

where U_r denotes the number of pins or vias contained in the G-cell c_r.

Definition 3.3.7 (Two-pin net)**.** A net that contains exactly two pins is referred to as a *two-pin net*.

Definition 3.3.8 (Multi-pin net)**.** A net containing more than two pins is termed a *multi-pin net*.

Definition 3.3.9 (Flat net)**.** A two-pin net N_i is classified as a *flat net* when its pins $P_{i,1}$ and $P_{i,2}$ reside in G-cells corresponding to vertices v_r and v_s, and these G-cells are located in the same row or column of the G-cell grid.

Definition 3.3.10 (Bend net)**.** A two-pin net N_i is identified as a *bend net* when its pins $P_{i,1}$ and $P_{i,2}$ occupy G-cells corresponding to vertices v_r and v_s that are not in the same row or column, and several possible paths made up of horizontal and vertical segments can connect the pins in v_r and v_s.

Table 3.1 presents the symbols used and their corresponding definitions.

Table 3.1: List of Symbols and Their Meanings

Symbols	**Meaning**
N	Netlist
N_i	i^{th} Net
$G = (V, E)$	Grid graph G with vertex set V and edge set E
v_r	Vertex r
e_{rs}	Edge (r, s)
c_r	Grid cell
L	Vertical metal layers
C_{rs}	Routing capacity of edge (r, s)
C_{rs}^{Stitch}	Stitch-avoiding routing capacity of edge (r, s)
U_{rs}	Number of net segments passing through the G-cell boundary corresponding to edge e_{rs}
C_r^{Stitch}	Stitch-avoiding vertex capacity of vertex r
U_r	Number of pins or vias in G-cell c_r
$P_{i,j}$	j^{th} pin of net N_i

Continued on next page

Table 3.1 continued from previous page

Symbols	**Meaning**
$SP_{i,j}$	j^{th} Steiner point of net N_i
S	Set of locations of stitch-lines
S_i	i^{th} stitch-line
n	Total number of nets
l	Number of stitch-lines
Γ	Set of stitch-avoiding rectilinear paths
Γ_i	Stitch-avoiding Rectilinear path for net N_i
n_1	Number of Category 1 TPBNs and TPBSs
n_2	Number of Category 2 TPBNs and TPBSs
NET_k	The set of n_1 Category 1 and Category 2 TPBNs and TPBSs located within the k^{th} box
$X_{i,1}$ and $X_{i,2}$	Denote the two candidate L-shaped routing paths
$Y_{i,1}$ and $Y_{i,2}$	Denote the two candidate U-shaped routing paths
NET_k^{Flat}	The set of n_3 Category 1 TPFSs within the k^{th} box
γ_i	Net-specific base weight N_i
$MD(\cdot)$	Manhattan Distance
$FPins(\cdot)$	Number of fixed pins in net N_i
$\alpha_{i,j}$	Weights with respect to stitch-line j^{th} candidate path for net N_i
$w_{i,f}$	Weight for a pin $P_{i,f}$ of net N_i
E_{Box^k}	Grid graph edges present in the k^{th} box
V_{Box^k}	Grid graph vertices present in the k^{th} box
A_{ij}^{stitch}	Available stitch-avoiding routing resource of edge (i, j)
A_r^{stitch}	Available stitch-avoiding vertex capacity of vertex v_r
Seg_i	i^{th} segment
φ	Extra bend penalty
$\rho(\cdot)$	Stitch penalty
τ	Stitch unfriendly region
$\#SB^{SL}$	Number of Steiner branches on stitch-lines
$\#SP^{SL}$	Number of Steiner points on stitch-lines
$\#SB^{\tau}$	Number of Steiner branches in stitch-unfriendly region

Continued on next page

Table 3.1 continued from previous page

Symbols	Meaning
$\#SP^{\tau}$	Number of Steiner points in stitch-unfriendly region
HPWL	Half-perimeter wirelength
#TVOF	Total number of vertex capacity overflows
#MVOF	Maximum vertex overflow
WL	Wirelength
$\#VV$	Number of via violations
$\#SHP$	Number of short polygon violations
$\#VRV$	Number of vertical routing violations

3.3.2 Problem Formulation

The stitch-avoiding global routing problem, defined on a grid-graph framework and incorporating C_{rs}^{Stitch} and C_{r}^{Stitch}, can be expressed as follows:

Given as input,

- A *netlist* $\mathcal{N} = \{N_1, N_2, \ldots, N_n\}$
- *Locations of stitch-lines* $S = \{S_1, S_2, \ldots, S_l\}$
- *The routing grid graph* $G = (V, E)$

Find a collection of stitch-avoiding rectilinear routes $\Gamma = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_n\}$ for each net $N_i \in \mathcal{N}$, such that the total routing length is minimized and all stitch-avoiding global routing constraints as well as vertex-capacity conditions are satisfied, i.e.,

$$\textit{minimize} \quad \sum_{i=1}^{n} |\Gamma_i| \tag{3.1}$$

$$\text{subject to} \quad U_{rs} \leq C_{rs}^{\text{Stitch}} \quad \forall\, e_{rs} \in E \tag{3.2}$$

$$U_r \leq C_{r}^{\text{Stitch}} \quad \forall\, v_r \in V \tag{3.3}$$

where $|\Gamma_i|$ denotes the routing length of net N_i.

3.4 Overview of Our Proposed Global Router for MEBL

A modern global router that relies solely on non-uniform G-cell capacities cannot reliably decrease stitch-line violations, since these violations are influenced by the placement of stitch-lines, pins, Steiner nodes, vias, and routing segments. To guide the detailed routing stage toward fewer violations, these elements must be incorporated into the global routing process so that path assignments are made more effectively. Figure 3.1 illustrates the structure of our stitch-avoiding global routing approach, inspired by *BoxRouter* [CP06]. The method consists of the following two stages:

Phase I: Stitch-avoiding Steiner Minimum Tree (SSMT) Construction

Phase II: Multistage stitch-avoiding global routing

In Phase I, we first generate Rectilinear Steiner Minimum Trees (RSMTs) for all the nets. These RSMTs are then modified to avoid stitch-lines and form SSMTs. This step is crucial because the RSMT branches serve as the backbone of the global routing solution. If the base routing is not stitch-avoiding, it becomes challenging to eliminate stitch-line-induced violations in later phases.

During Phase II, the SSMTs of each net serve as the initial routing solution, from which the final global routes for flat, bend, and multi-pin nets are produced. The routing process aims to reduce overall wirelength while adhering to constraints for stitch-avoiding global routing and vertex-capacity limitations.

Section 3.5 and Section 3.6 provide a detailed description of Phase I (Stitch-Avoiding Steiner Minimum Tree construction) and Phase II (Multistage Stitch-Avoiding Global Routing) of the proposed methodology.

3.5 Phase-I: Stitch-avoiding Steiner Minimum Tree (SSMT)

For each multi-pin net, we begin by generating an initial Rectilinear Steiner Minimum Tree (RSMT) using FLUTE [CW07], a fast and accurate heuristic for constructing Steiner trees in rectilinear space. Although efficient, the RSMT may introduce design

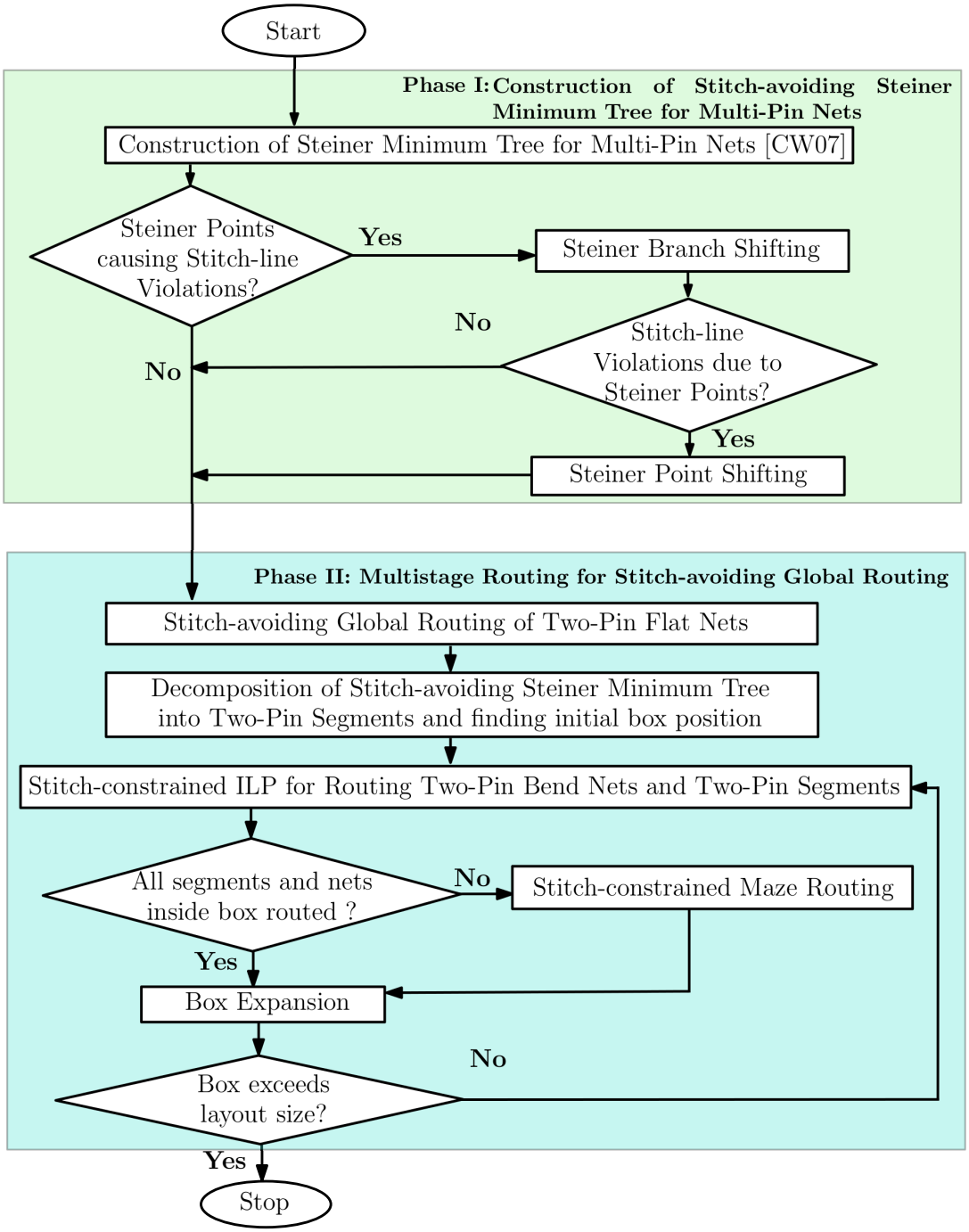


Figure 3.1: Overview of our proposed Global Routing Method for MEBL

rule violations due to the placement of Steiner points either along stitch-lines—leading to via or vertical routing violations—or within stitch-unfriendly regions, which can cause short polygon violations. These violations compromise the feasibility of the routing solution in downstream stages. To mitigate these issues, we introduce the following two targeted shifting mechanisms:

- Steiner Branch Shifting (SBS)
- Steiner Point Shifting (SPS)

Although prior work, such as [CLYP09], has used similar shifting techniques for congestion-sensitive RSMT generation, our approach specifically focuses on mitigating stitch-line-induced violations. A Steiner branch or point is shifted only if its position introduces a stitch-related violation.

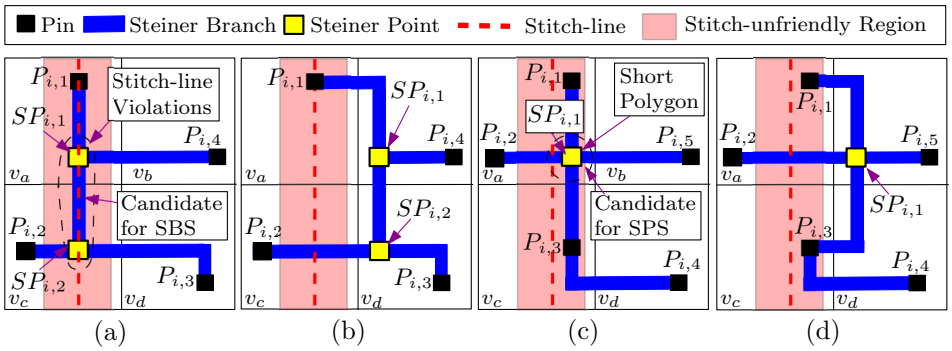


Figure 3.2: (a) Steiner branch between $SP_{i,1}$ and $SP_{i,2}$ causing via and vertical routing violations, (b) violations removed after SBS, (c) Steiner point $SP_{i,1}$ causing short polygon violation, and (d) short polygon removed after SPS.

In SBS, a branch of the RSMT is considered for shifting when it joins two Steiner nodes and is fully contained within a stitch-unfriendly area, as illustrated in Figure 3.2(a) and Figure 3.2(b). For SPS, a Steiner node is eligible for shifting when it either lies on a stitch-line or is located inside a stitch-unfriendly zone, and at least one of its adjacent branches intersects a stitch-line, as depicted in Figure 3.2(c) and Figure 3.2(d).

Because stitch-lines are considered vertical, both SBS and SPS conduct horizontal scans to determine feasible shift positions. In SBS, the method initially looks for a routing area with adequate capacity just beyond the closest edge of the stitch-

unfriendly zone. If that fails, the search continues toward the opposite side. When no region with enough capacity is available, the stitch-related conflict cannot be eliminated at the global routing stage. For SPS, a horizontal line-probing strategy is applied to check potential relocation points, aiming to move the Steiner node to the nearest area that is both stitch-permissive and free from congestion.

To illustrate the proposed stitch-avoiding RSMT modification approach, consider a multi-pin net as shown in Figure 3.3(a) with terminals located at coordinates $P_{i,1} = (3, 5)$, $P_{i,2} = (8, 18)$, $P_{i,3} = (11, 3)$, and $P_{i,4} = (18, 11)$. The vertical stitch-line is assumed to lie along $x = 8$, and the stitch-unfriendly region spans from $x = 7$ to $x = 9$. Using FLUTE [CW07], an initial RSMT is generated with Steiner points $SP_{i,1} = (8, 11)$ and $SP_{i,2} = (8, 5)$, connecting the terminals with minimum wirelength.

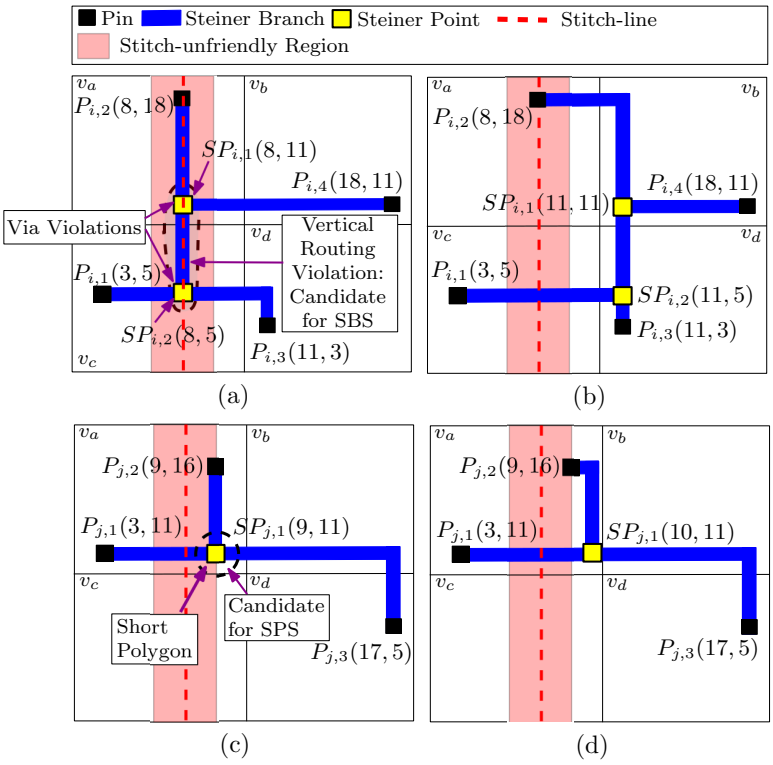


Figure 3.3: (a) A Steiner branch connecting $SP_{i,1}$ and $SP_{i,2}$ leads to via-related and vertical routing conflicts, (b) these conflicts are eliminated after applying SBS, (c) Steiner point $SP_{j,1}$ introduces a short-polygon violation, and (d) this violation is resolved through SPS with an added wirelength of one unit.

However, as Figure 3.3(a) shows, the RSMT introduces several stitch-induced violations. First, the branch connecting $SP_{i,1}$ and $SP_{i,2}$ lies on the stitch-line at $x = 8$. Since this branch is vertical, it results in a *vertical routing violation*, which is considered a *hard constraint* due to the risk of severe pattern distortion caused by overlay errors during manufacturing.

Additionally, the Steiner points $SP_{i,1}$ and $SP_{i,2}$ lie directly on the stitch-line and will be connected to a via and will cause a via violation. To resolve these violations, we apply the proposed shifting techniques. Using *Steiner Branch Shifting (SBS)*, the branch between $SP_{i,1}$ and $SP_{i,2}$ is horizontally shifted to the right, relocating $SP_{i,1}$ to $(11, 11)$ and $SP_{i,2}$ to $(11, 5)$ as shown in Figure 3.3(b). This eliminates both the via and vertical routing violations by moving the branch outside the stitch-unfriendly zone and away from the stitch-line. The resulting RSMT maintains a similar total wirelength while satisfying all stitch-related constraints, producing a more manufacturable and lithography-friendly global routing solution.

Subsequently, in Figure 3.3(c), we have another RSMT with $P_{j,1} = (3, 11)$, $P_{j,2} = (9, 16)$, $P_{j,3} = (17, 5)$, and Steiner point $SP_{j,1} = (9, 11)$ causing a short polygon violation at $SP_{j,1}$. *Steiner Point Shifting (SPS)* is used to relocate $SP_{j,1}$ from $(9, 11)$ to $(10, 11)$ using a horizontal line-probing method as shown in Figure 3.3(d). This shift places $SP_{j,1}$ in a stitch-friendly region and eliminates the short polygon violation by avoiding a leading via placement on the stitch-line. However, it increases the wirelength by 1 unit.

3.6 Phase-II: Multistage Stitch-avoiding Global Routing

After constructing the SSMTs, we schedule the nets for global routing by first handling the two-pin flat nets, which offer the least routing flexibility, then the two-pin bend nets, and finally the multi-pin nets.

3.6.1 Stitch-avoiding Global Routing of Two-Pin Flat Nets

Two-pin flat nets are characterized by having only a single shortest path connecting their two terminals, which spans either entirely horizontally or entirely vertically. Due to the lack of path flexibility, these nets must be routed early and efficiently to avoid routing conflicts. These nets are routed by applying a shortest-path search on the routing grid graph G, where each edge's cost is derived from its available stitch-aware capacity A_{ij}^{stitch}. This allows the algorithm to favor paths with higher remaining capacity and lower likelihood of congestion.

However, special consideration is required for vertically spanned two-pin flat nets that intersect stitch-holding cells. These regions are sensitive to stitch-induced constraints and require careful resource management to avoid violating lithographic rules. To handle such cases, we employ a dedicated stitch-constrained maze routing algorithm, which accounts for both the stitch-line positions and the stitch-unfriendly regions within the routing space. This ensures that routing through stitch-holding cells remains feasible without introducing via or vertical segment violations. The complete details of this routing method are presented in Section 3.6.3.

3.6.2 Stitch-avoiding Global Routing for Multi-Pin Nets and TPBN

For every multi-pin net, its SSMT is broken down into several *two-pin segments (TPSs)*. Each such segment is assigned to one of the following two classes according to its geometric form and routing behavior:

- *Two-Pin Flat Segment* (TPFS)
- *Two-Pin Bend Segment* (TPBS)

Following the methodology of *BoxRouter* [CP06], we define a box over the layout that encompasses a group of adjacent G-cells. In our approach, the initial box is formed by selecting the four most densely populated contiguous G-cells.

Inside this region, we first solve a *stitch-aware Integer Linear Programming (ILP)* model to assign routes optimally for the TPBSs and TPSs contained within it. Afterward, *stitch-aware maze routing* is applied to construct the actual paths while

respecting stitch constraints. The region is then gradually enlarged to incorporate more G-cells along with the nets intersecting them. During each expansion step, routing is carried out only for the newly added nets and any nets left unresolved, following the same two-stage approach: an ILP considering stitch restrictions, and then maze routing subject to the same limits. This enlargement proceeds until the full layout is processed, resulting in a complete global routing solution that avoids stitch violations.

At each iteration, the routing is performed solely for TPBNs and TPSs (both TPFSs and TPBSs) that lie within the current box. This localized routing keeps the ILP formulation compact, ensuring that its solving remains computationally efficient. Additionally, box expansion can be adaptively controlled based on net density in the layout, allowing the routing process to scale gracefully with circuit complexity.

The ILP incorporates several constraints and weight parameters to generate a routing solution that minimizes stitch-line violations. During the routing of TPBNs and TPSs within the k^{th} box, the nets and segments are categorized into the following three groups:

- *Category 1:* TPBNs and TPSs newly introduced in the k^{th} box.
- *Category 2:* TPBNs and TPSs that remained unrouted in the ILP solution for the $(k-1)^{\text{th}}$ box.
- *Category 3:* TPBNs and TPSs that were unrouted in both the $(k-1)^{\text{th}}$ and $(k-2)^{\text{th}}$ boxes.

Let NET_k denote the set of n_1 Category 1 and n_2 Category 2 TPBNs and TPBSs located within the k^{th} box. Similarly, let NET_k^{Flat} represent the set of n_3 Category 1 TPFSs within the same box.

As illustrated in Figure 3.4(a), for *Category 1*, only two L-shaped routing patterns are considered for each TPBN and TPBS to minimize routing bends, which can introduce additional vias and potentially lead to increased stitch-line violations. Let $X_{i,1}$ and $X_{i,2}$ denote the two candidate L-shaped routing patterns for net N_i. For TPFSs, only a single horizontal or vertical routing pattern is considered, denoted by $X_{i,1}$.

In the proposed stitch-constrained ILP formulation, weights are assigned to each routing pattern $X_{i,1}$ and $X_{i,2}$ within the objective function to guide the solver to-

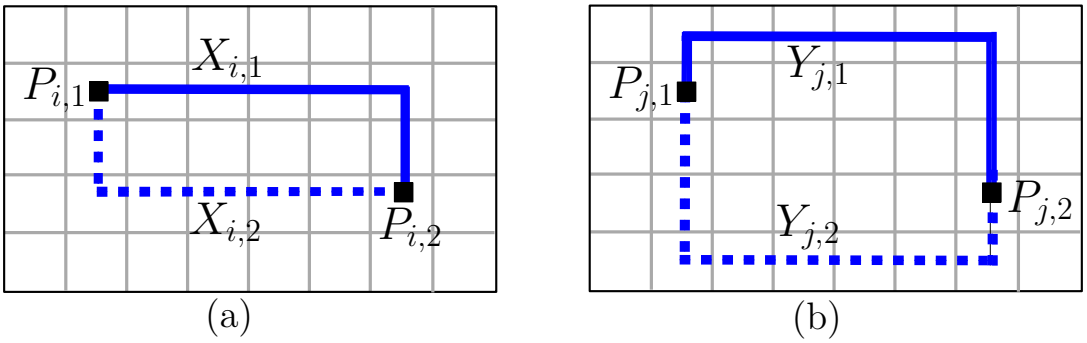


Figure 3.4: Illustration of two patterns corresponding to a net: (a) *Category* 1, (b) *Category* 2

ward solutions that reduce stitching violations. Section 3.6.2.1 describes the weight assignment depending on different cases.

3.6.2.1 Weight Assignment Based on Net Length and Type

Small nets typically offer less routing flexibility and should therefore be prioritized during the routing process. To capture this behavior, we incorporate the Manhattan Distance (MD) between the endpoints of each net into the weight function. Among the net categories, TPBNs are the least flexible since both of their endpoints are fixed pins, while TPSs offer more routing freedom. As a result, TPBNs are assigned a higher weight. The weight γ_i for a net N_i, which reflects both its length and category, is defined in Equation 3.4. Here, $\text{FPins}(N_i)$ denotes the number of fixed pins in net N_i, and the ratio term promotes shorter and less flexible nets by assigning them higher weights.

$$\gamma_i = \left\lceil \frac{\sum_{N_j \in NET_k \cup NET_k^{Flat}} \text{MD}(N_j)}{(n_1 + n_2 + n_3) \cdot \text{MD}(N_i)} \right\rceil + \text{FPins}(N_i) \tag{3.4}$$

To demonstrate the weight computation for different nets and segments, consider a routing stage where we have the following nets and segments:

- Two TPBNs: N_1 (Category 1) and N_2 (Category 2)
- Two TPBSs: Seg_1 (Category 1) and Seg_2 (Category 2)
- One TPFS: Flt_1 (Category 1)

The Manhattan distances between the endpoints and the number of fixed pins for each are summarized in the Table 3.2. Here, in the example, $n_1 = 2$ (includes N_1 and Seg_1), $n_2 = 2$ (includes N_2 and Seg_2) and $n_3 = 1$ (includes Flt_1).

Table 3.2: Net and Segment Data for Weight Calculation

Net/Segment	**Type**	**Category**	**MD (units)**	**FPins**
N_1	TPBN	1	10	2
N_2	TPBN	2	20	2
Seg_1	TPBS	1	15	1
Seg_2	TPBS	2	30	1
Flt_1	TPFS	1	12	1

The total Manhattan distance of all nets/segments is:

$$\sum_{N_j \in NET_k \cup NET_k^{Flat}} \text{MD}(N_j) = 10 + 20 + 15 + 30 + 12 = 87$$

We now compute the weight γ_i for each net/segment:

- For N_1:

$$\gamma_1 = \left\lceil \frac{87}{(2+2+1)\cdot 10} \right\rceil + 2 = \left\lceil \frac{87}{50} \right\rceil + 2 = \lceil 1.74 \rceil + 2 = 2 + 2 = 4$$

- For N_2:

$$\gamma_2 = \left\lceil \frac{87}{(2+2+1)\cdot 20} \right\rceil + 2 = \left\lceil \frac{87}{100} \right\rceil + 2 = \lceil 0.87 \rceil + 2 = 1 + 2 = 3$$

- For Seg_1:

$$\gamma_3 = \left\lceil \frac{87}{(2+2+1)\cdot 15} \right\rceil + 1 = \left\lceil \frac{87}{75} \right\rceil + 1 = \lceil 1.16 \rceil + 1 = 2 + 1 = 3$$

- For Seg_2:

$$\gamma_4 = \left\lceil \frac{87}{(2+2+1)\cdot 30} \right\rceil + 1 = \left\lceil \frac{87}{150} \right\rceil + 1 = \lceil 0.58 \rceil + 1 = 1 + 1 = 2$$

- For Flt_1:

$$\gamma_5 = \left\lceil \frac{87}{(2+2+1)\cdot 12} \right\rceil + 1 = \left\lceil \frac{87}{60} \right\rceil + 1 = \lceil 1.45 \rceil + 1 = 2 + 1 = 3$$

The computed weights reflect the relative priority of the nets and segments during the routing process. Since N_1 is a short TPBN with two fixed pins, it gets the highest priority ($\gamma = 4$). In contrast, longer segments or more flexible types like Seg_2 (TPBS of category 2) receive a lower weight, indicating they can be routed later without much congestion risk.

3.6.2.2 Weight Calculation Considering Routing Congestion and Endpoint Positions Relative to Stitch-Lines

To enhance routability, we assign a weight to each candidate path $X_{i,f}$ (where $f = 1$ or 2) based on the ratio of the minimum available capacity (MinAvailableCapacity) to the maximum demand (MaxDemand) along the path. This weighting encourages routing through less congested regions. Furthermore, the relative position of the endpoints of Two-Pin Bend Nets (TPBNs) and Two-Pin Bend Segments (TPBSs) with respect to stitch-lines influences the priority assigned to each path. The priority of a candidate path $X_{i,f}$ is denoted by $\alpha_{i,f}$, where $f = 1$ or 2, and is determined according to the stitch-line alignment of its endpoints.

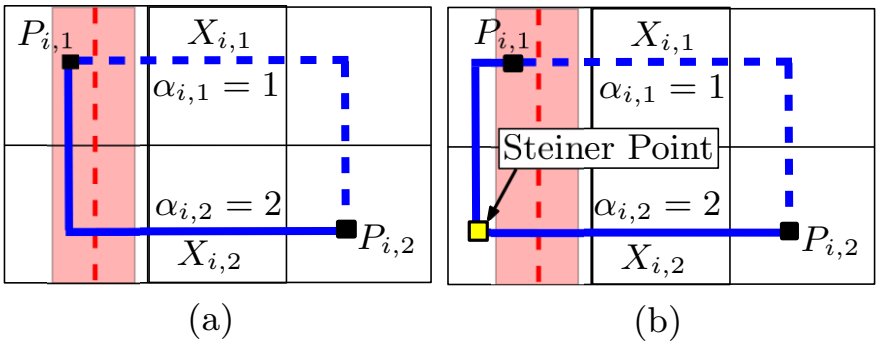


Figure 3.5: (a) Computation of $\alpha_{i,1}$ and $\alpha_{i,2}$ based on endpoint position relative to stitch-line; (b) Making path $X_{i,2}$ violation-free using an additional Steiner point

Algorithm 3.1: Finding $\alpha_{i,1}$ and $\alpha_{i,2}$ for each net N_i

Input : NET_k: Set of TPBNs and TPBSs inside box k

Output: Weights $\alpha_{i,1}, \alpha_{i,2}$ for all $N_i \in NET_k$

1 **for** *each net* $N_i \in NET_k$ **do**

  // $\tau Check(P_{i,f})$ returns true if $P_{i,f}$ is in a stitch-unfriendly region

2 $t_1 \leftarrow \tau Check(P_{i,1})$;

3 $t_2 \leftarrow \tau Check(P_{i,2})$;

  // $xStitch(P_{i,f})$ gives x-coordinate of stitch-line of pin $P_{i,f}$'s cell

4 $S_1 \leftarrow xStitch(P_{i,1})$;

5 $S_2 \leftarrow xStitch(P_{i,2})$;

6 **if** $\Big(t_1 \wedge t_2 \wedge (x(P_{i,1}) < S_1) \wedge (x(P_{i,2}) < S_2)\Big) \vee \Big(t_1 \wedge \neg t_2 \wedge (x(P_{i,1}) < S_1)\Big) \vee \Big(\neg t_1 \wedge t_2 \wedge (x(P_{i,2}) < S_2)\Big)$ **then**

7 $\alpha_{i,1} \leftarrow 1$, and $\alpha_{i,2} \leftarrow 2$; // Case I

8 **else if** $\Big(t_1 \wedge t_2 \wedge (x(P_{i,1}) > S_1) \wedge (x(P_{i,2}) > S_2)\Big) \vee \Big(t_1 \wedge \neg t_2 \wedge (x(P_{i,1}) > S_1)\Big) \vee \Big(\neg t_1 \wedge t_2 \wedge (x(P_{i,2}) > S_2)\Big)$ **then**

9 $\alpha_{i,1} \leftarrow 2$, and $\alpha_{i,2} \leftarrow 1$; // Case II

10 **else**

11 $\alpha_{i,1} \leftarrow 1$, and $\alpha_{i,2} \leftarrow 1$; // Case III

12 **end**

13 **end**

Let $P_{i,1}$ and $P_{i,2}$ denote the two endpoints of the net N_i. Based on the relative positions of the stitch-lines with respect to these pin locations, Algorithm 3.1 determines the priority values $\alpha_{i,1}$ and $\alpha_{i,2}$ for TPBNs and TPBSs. If $\alpha_{i,1}$ is greater than $\alpha_{i,2}$, then path $X_{i,1}$ is considered more favorable than $X_{i,2}$ in the presence of stitch-lines, and vice versa. For TPFSs, only a single path is evaluated, so no path preference applies. Therefore, we set $\alpha_{i,1} = 1$ for all TPFSs.

Figure 3.5(a) illustrates a scenario that corresponds to Case I of Algorithm 3.1. It presents two candidate L-shaped paths for net N_i, labeled as $X_{i,1}$ and $X_{i,2}$. The

positional relationship between the pin and stitch-line satisfies the condition $t_1 = \text{true} \wedge t_2 = \text{false} \wedge x(P_{i,1}) < S_1$, where $S_1 = xStitch(P_{i,1})$.

If path $X_{i,1}$ is selected, it results in the formation of a short polygon at $P_{i,1}$. This leads to one of the following two scenarios:

- *Fixed pin:* If $P_{i,1}$ is a fixed pin, it is inherently unmovable.
- *Steiner point:* If $P_{i,1}$ is a Steiner point located within the stitch-unfriendly region, this indicates that both the SBS and SPS algorithms failed to relocate it outside the region due to insufficient available space. Consequently, these types of Steiner points can also be treated as unmovable.

Hence, regardless of whether $P_{i,1}$ is a fixed pin or an immovable Steiner point, the short polygon formed at $P_{i,1}$ is unavoidable.

On the other hand, if we select path $X_{i,2}$, no short polygon is formed at $P_{i,1}$. However, a via may be introduced during detailed routing at the junction of the horizontal and vertical segments of $X_{i,2}$. If the vertical segment lies within the stitch-unfriendly region, this via may lead to a short polygon violation. To proactively mitigate this issue during the global routing stage, a Steiner point can be introduced outside the stitch-unfriendly region, as illustrated in Figure 3.5(b), thereby avoiding any potential violation. As a result, we assign $\alpha_{i,1} = 1$ and $\alpha_{i,2} = 2$, giving higher priority to path $X_{i,2}$ over $X_{i,1}$.

To illustrate the behavior of Algorithm 3.1, consider a box k containing the following nets: N_1, N_2, N_3, and N_4. Each net N_i has two pins $P_{i,1}$ and $P_{i,2}$, with known x-coordinates $x(P_{i,1})$ and $x(P_{i,2})$, stitch-line coordinates S_1 and S_2, and Boolean flags t_1 and t_2 indicating whether each pin lies in a stitch-unfriendly region, as determined by the function $\tau Check(P_{i,f})$. The weights $\alpha_{i,1}$ and $\alpha_{i,2}$ are then assigned based on the rules defined by Algorithm 3.1. Table 3.3 summarizes the configuration and resulting weight assignments for each net:

In net N_1, only the first pin lies in a stitch-unfriendly region and it is located to the left of its corresponding stitch-line ($x(P_{1,1}) < S_1$), matching *Case I*. Hence, $\alpha_{1,1} = 1$ and $\alpha_{1,2} = 2$. Net N_2 corresponds to *Case II* since the second pin is in a stitch-unfriendly region and is located to the right of its stitch-line ($x(P_{2,2}) > S_2$), leading to $\alpha_{2,1} = 2$ and $\alpha_{2,2} = 1$. Net N_3 also satisfies the condition for *Case II*,

Table 3.3: Example of assigning weights $\alpha_{i,1}$ and $\alpha_{i,2}$ using Algorithm 3.1, based on the position of pins and stitch-lines. The table covers all three cases handled by the algorithm.

Net	t_1	t_2	$x(P_{i,1})$	$x(P_{i,2})$	S_1	S_2	Case	$\alpha_{i,1}$	$\alpha_{i,2}$
N_1	true	false	3	8	5	10	I	1	2
N_2	false	true	4	9	6	8	II	2	1
N_3	true	true	12	14	10	13	II	2	1
N_4	false	false	6	7	9	10	III	1	1

as both pins lie in stitch-unfriendly regions and are positioned to the right of their respective stitch-lines. Thus, it also receives $\alpha_{3,1} = 2$ and $\alpha_{3,2} = 1$. Finally, net N_4 falls into *Case III*, where neither pin is in a stitch-unfriendly region. Consequently, both weights are set to 1: $\alpha_{4,1} = \alpha_{4,2} = 1$.

This example comprehensively demonstrates how the algorithm assigns asymmetric weights based on pin location and stitch-unfriendly constraints, guiding downstream optimization steps accordingly.

The weight $w_{i,f}$ for a pin $P_{i,f}$ of net N_i is computed using Equation 3.5, which combines the net-specific base weight γ_i with a congestion-aware penalty term scaled by the weight coefficient $\alpha_{i,f}$. The second term evaluates the local congestion around the pin based on the minimum available capacity and maximum demand in the vicinity of $X_{i,f}$.

$$w_{i,f} = \gamma_i + \alpha_{i,f} \left\lceil \frac{MinAvailableCapacity(X_{i,f})}{MaxDemand(X_{i,f})} \right\rceil \tag{3.5}$$

The inner fraction compares the *minimum available routing capacity* to the *maximum expected demand* within the congestion window centered at $X_{i,f}$. This ratio effectively captures the local congestion level—a lower value implies higher congestion. The ceiling function is applied to ensure that the computed ratio is rounded up, preventing underestimation of congestion. The coefficient $\alpha_{i,f}$, determined by the algorithm in Algorithm 3.1, biases the weight based on the relative placement of the pin with respect to stitch-lines and stitch-unfriendly regions. For example, a pin closer to the left of a stitch-line in a stitch-unfriendly region may receive a higher

weight coefficient to discourage its inclusion in routing. The base weight γ_i can reflect other factors such as net criticality or net length.

Thus, this formulation ensures that the weight $w_{i,f}$ not only reflects the static characteristics of a net but also dynamically adapts based on local routing congestion and stitch-aware placement considerations, thereby promoting more balanced and manufacturable routing decisions.

3.6.2.3 ILP-Based Stitch-Constrained Routing in the k^{th} Box

Category 2 TPBNs and TPBSs that remain unrouted after the $(k-1)^{\text{th}}$ iteration indicate that the available routing resources are insufficient for completing their connections using only L-shaped patterns. Consequently, additional bends are required to complete these routes. To address this, we consider two alternative routing patterns with two bends, as illustrated in Figure 3.4(b). These two-bend paths, denoted by $Y_{j,1}$ and $Y_{j,2}$ for net N_j, incur a wirelength overhead compared to the Category 1 TPBNs and TPBSs. To reflect this disadvantage, a penalty factor φ is subtracted from the weights of $Y_{j,1}$ and $Y_{j,2}$.

Nets falling under Category 3 are not considered for routing within the k^{th} box. Instead, they are deferred and handled later using the stitch-constrained maze routing approach described in Section 3.6.3.

The generalized stitch-constrained Integer Linear Program (ILP) used to route the nets within the k^{th} box is presented in Figure 3.6. Let E_{Box^k} and V_{Box^k} denote the sets of edges and vertices of graph G that lie within the k^{th} box, respectively. The ILP aims to maximize the number of successfully routed nets by selecting appropriate routing candidates for various net types, while ensuring that all stitching and resource constraints are satisfied.

The objective function in Equation 3.6 seeks to maximize the total routing weight for all net types considered. This includes: (i) n_1 two-pin bend nets (TPBNs) introduced in the current iteration, where routing options are denoted by variables $X_{i,1}$ and $X_{i,2}$; (ii) n_2 TPBNs carried over from the previous iteration that could not be routed using L-shaped patterns and are now assigned two-bend options, denoted by variables $Y_{j,1}$ and $Y_{j,2}$, each penalized by a factor φ; and (iii) n_3 two-pin flat segments

$$\max \sum_{i=1}^{n_1}\sum_{f=1}^{2} w_{i,f}X_{i,f} + \sum_{j=1}^{n_2}\sum_{q=1}^{2}(w_{j,q}-\varphi)Y_{j,q} + \sum_{l=1}^{n_3} w_{l,1}X_{l,1} \tag{3.6}$$

subject to:

$$X_{i,1} + X_{i,2} \leq 1 \qquad \forall N_i \in (NET_k \backslash NET_{k-1}) \tag{3.7}$$

$$Y_{j,1} + Y_{j,2} \leq 1 \qquad \forall N_j \in (NET_k \cap NET_{k-1}) \tag{3.8}$$

$$X_{l,1} \leq 1 \qquad \forall N_l \in NET_k^{Flat} \tag{3.9}$$

for each $e_{rs} \in E_{Box^k}$, $X_{i,f}, Y_{i,q}$ intersects e_{rs}

$$\sum_{e_{rs}} (X_{i,f} + Y_{i,q}) \leq A_{rs}^{stitch} \tag{3.10}$$

for each $v_r \in V_{Box^k}$, Bend of $X_{i,f}, Y_{i,q}$ lies in v_r

$$\sum_{v_r} (X_{i,f} + Y_{i,q}) \leq A_{r}^{stitch} \tag{3.11}$$

$$X_{i,1}, X_{i,2}, Y_{j,1}, Y_{j,2} \in \{0, 1\} \tag{3.12}$$

Figure 3.6: Generalized stitch-constrained ILP to route two-pin bend nets and two-pin segments present inside the k^{th} box

(TPFSs), where each segment may be routed using a single candidate denoted $X_{l,1}$.

Constraint in Equation 3.7 ensures that for every Category 1 TPBN $N_i \in NET_k \backslash NET_{k-1}$, at most one of the two candidate paths $X_{i,1}$ or $X_{i,2}$ is selected. This prevents redundant routing for the same net. Similarly, Constraint in Equation 3.8 enforces the same exclusivity for Category 2 TPBNs $N_j \in NET_k \cap NET_{k-1}$ by allowing at most one among $Y_{j,1}$ and $Y_{j,2}$ to be selected. For each flat TPFS $N_l \in NET_k^{\text{Flat}}$, Constraint in Equation 3.9 ensures that at most one of its candidate routing paths X_l is selected.

Routing resource constraints are introduced next. Constraint in Equation 3.10 enforces *edge stitch capacity*: it ensures that the cumulative usage of any routing edge $e_{rs} \in E_{\text{Box}^k}$ across all nets does not exceed the stitch resource limit A_{rs}^{stitch}. Constraint in Equation 3.11 imposes *vertex stitch capacity*: it restricts the number of bends or stitch points placed on any vertex $v_r \in V_{\text{Box}^k}$ to remain within its capacity A_r^{stitch}.

Finally, Constraint Equation 3.12 enforces the *integrality condition*, ensuring that all decision variables ($X_{i,1}$, $X_{i,2}$, $Y_{j,1}$, $Y_{j,2}$, X_l) are binary-valued. That is, each variable can take a value of either 0 or 1, corresponding to whether a specific routing pattern is chosen or not.

To illustrate the working of the generalized stitch-constrained ILP shown in Figure 3.6, we consider a simplified example with three nets inside the k^{th} box: one two-pin bend net N_1 that appears only in the k^{th} box (i.e., $NET_k \backslash NET_{k-1}$), one two-pin segment net N_2 that spans both $k-1^{th}$ and k^{th} boxes (i.e., $NET_k \cap NET_{k-1}$), and one flat net N_3 that is considered part of NET_k^{Flat}. For this example, assume the following: In this example, all routing edges e_{rs} are assigned a unit capacity, i.e., $A_{rs}^{stitch} = 1$ for all edges e_{rs}. Similarly, each routing vertex v_r also has a capacity of one, meaning $A_r^{stitch} = 1$ for all vertices v_r. The weights associated with each candidate path are as follows: for net N_1, $w_{1,1} = 4$ and $w_{1,2} = 5$; for net N_2, $w_{2,1} = 6$ and $w_{2,2} = 4$; and for net N_3, only one path exists with weight $w_{3,1} = 3$. A stitch penalty of $\varphi = 2$ is applied for any additional bend-based pattern. With regard to intersections, both candidate paths $X_{1,1}$ and $X_{1,2}$ of net N_1 intersect the edge e_{12} and bend at the vertex v_1. For net N_2, path $Y_{2,1}$ intersects edge e_{12} and bends at v_1, while path $Y_{2,2}$ intersects edge e_{23} and bends at v_2. Finally, the candidate path $X_{3,1}$ of net N_3 intersects edge e_{12} and bends at vertex v_1. The ILP formulation for this instance becomes:

Equation 3.13 defines the objective: maximize the total gain from selecting net segments, while penalizing stitch segments (via φ). The constraints in Equation 3.14 – Equation 3.16 ensure that each net uses at most one configuration (either bend or segment). Constraint in Equation 3.17 and Equation 3.18 enforce edge capacity limits, preventing multiple overlapping net segments on the same edge. The constraints in Equation 3.19 and Equation 3.20 similarly limit the bends through a single vertex. Finally, constraint in Equation 3.21 enforces binary decisions.

The optimal solution to the ILP is obtained by selecting $X_{1,2} = 1$, $Y_{2,2} = 1$, and $X_{3,1} = 0$. In this configuration, the selected path $X_{1,2}$ contributes a value of 5 to the objective function. The selected path $Y_{2,2}$ contributes 2, which is computed as $w_{2,2} - \varphi = 4 - 2 = 2$, due to the stitch penalty applied. The path $X_{3,1}$ is not selected, and therefore contributes 0 to the objective. As a result, the total value of

$$\max \quad 4X_{1,1} + 5X_{1,2} + (6-2)Y_{2,1} + (4-2)Y_{2,2} + 3X_{3,1} \tag{3.13}$$

subject to:

$$X_{1,1} + X_{1,2} \leq 1 \tag{3.14}$$

$$Y_{2,1} + Y_{2,2} \leq 1 \tag{3.15}$$

$$X_{3,1} \leq 1 \tag{3.16}$$

$$X_{1,1} + X_{1,2} + Y_{2,1} + X_{3,1} \leq 1 \quad \text{(for edge } e_{12}\text{)} \tag{3.17}$$

$$Y_{2,2} \leq 1 \quad \text{(for edge } e_{23}\text{)} \tag{3.18}$$

$$X_{1,1} + X_{1,2} + Y_{2,1} + X_{3,1} \leq 1 \quad \text{(for vertex } v_1\text{)} \tag{3.19}$$

$$Y_{2,2} \leq 1 \quad \text{(for vertex } v_2\text{)} \tag{3.20}$$

$$X_{1,1}, X_{1,2}, Y_{2,1}, Y_{2,2}, X_{3,1} \in \{0, 1\} \tag{3.21}$$

Figure 3.7: Example of our Stitch-constrained ILP with three nets inside the kth box

the objective function becomes $5 + 2 = 7$.

Choosing $X_{1,2}$ and $Y_{2,2}$ satisfies all constraints. Both intersect different edges (e_{12} and e_{23}) and vertices (v_1 and v_2), so the capacity constraints are respected. Including $X_{3,1}$ would violate either edge or vertex capacity on e_{12} and v_1. Thus, this solution maximizes gain while satisfying all stitch-related constraints.

Complexity Analysis of the Proposed ILP Formulation As discussed in Section 3.6.2, our proposed stitch-avoiding global routing framework employs a stitch-constrained 0–1 Integer Linear Programming (ILP) formulation and therefore inherits the NP-hard nature of combinatorial routing optimization problems. A direct global ILP formulation over the entire routing layout would become computationally prohibitive as the number of nets, routing resources, and stitch-related constraints increase.

To address this challenge, as discussed in Section 3.6.2, the proposed methodology adopts a progressive box-expansion strategy inspired by *BoxRouter*. Rather than solving a single large optimization problem, routing is performed through a sequence of localized ILP instances. During each iteration, only the TPBNs, TPSs, and previously unrouted segments contained within the current routing box are considered. As

stated in this chapter, "only the TPBNs and TPSs present inside the current box are routed, so the size of the ILP is small." Furthermore, the amount of box expansion can be controlled according to the routing density of the design.

For the k^{th} routing box, let n_1, n_2, and n_3 denote the numbers of Category-1 TPBNs/TPSs, Category-2 TPBNs/TPSs, and TPFSs, respectively. From the ILP formulation, each Category-1 net contributes two binary decision variables (X_i^1, X_i^2), each Category-2 net contributes two binary decision variables (Y_j^1, Y_j^2), and each TPFS contributes one binary decision variable (X_l^1). Consequently, the total number of binary variables in the ILP is

$$m_k = 2n_1 + 2n_2 + n_3.$$

Since the formulation is a 0–1 ILP, the worst-case complexity of the branch-and-bound search performed by the ILP solver is exponential in the number of binary variables. Therefore, the worst-case complexity of solving the ILP associated with the k^{th} routing box is

$$T_{\text{ILP}}(k) = O(2^{m_k}) = O\left(2^{(2n_1+2n_2+n_3)}\right).$$

After the ILP stage, the unrouted TPBNs, TPSs, and two-pin flat nets are routed using the proposed stitch-constrained maze router based on Hadlock's algorithm. For a routing box containing E_k routing edges and r_k unrouted nets, the maze-routing phase requires

$$T_{\text{Maze}}(k) = O(r_k E_k).$$

Therefore, the overall complexity of routing the k^{th} box can be expressed as

$$T_{\text{Box}}(k) = O\left(2^{(2n_1+2n_2+n_3)}\right) + O(r_k E_k).$$

Since the ILP component dominates the computational cost, the complexity is primarily governed by

$$T_{\text{Box}}(k) = O\left(2^{(2n_1+2n_2+n_3)}\right).$$

If the routing process generates B progressively expanded routing boxes, the overall complexity of the proposed framework is

$$T_{\text{Total}} = \sum_{k=1}^{B} O(2^{m_k}) + \sum_{k=1}^{B} O(r_k E_k),$$

where

$$m_k = 2n_{1k} + 2n_{2k} + n_{3k}.$$

Thus,

$$T_{\text{Total}} = O\left(\sum_{k=1}^{B} 2^{m_k}\right),$$

which is significantly smaller in practice than solving a single global ILP because each m_k depends only on the routing objects contained within the current box rather than the entire design.

3.6.3 Stitch-constrained Maze Routing

After solving the ILP for the k^{th} box, the previously unrouted two-pin flat nets, as well as the Category 3 two-pin bend nets (TPBNs) and two-pin segments (TPSs), are routed. Since these nets are often more complex and require more detours, additional bends may be introduced during their routing. To handle this efficiently, we employ a *stitch-constrained Maze Router* based on Hadlock's algorithm [Had75], which uses the concept of *detour number*—defined as the number of grid cells that lie in the direction away from the target cell.

In our proposed method, for each vertex $v \in G$, the routing decision does not rely solely on the detour numbers of its neighboring vertices. Instead, we define a cost function that incorporates the following four key parameters:

- The *detour number*
- The *available routing resource* on the edge between v and its neighbor
- The *available vertex capacity*

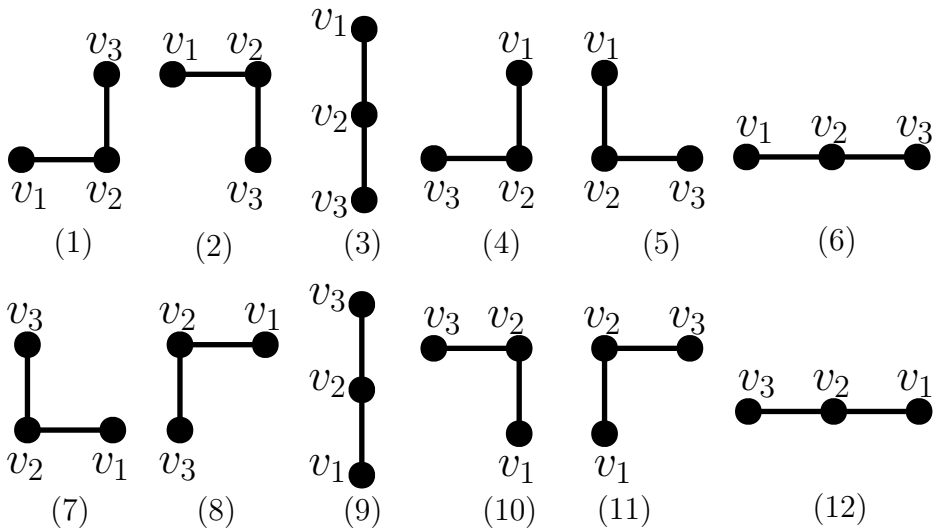


Figure 3.8: Twelve possible patterns considered during Retrack

- *Stitch Penalty*: The Stitch Penalty $\rho(v_r, v_s)$ is applied only when both G-cells c_r and c_s are designated stitch-holding cells; in this case, $\rho(v_r, v_s)$ is equal to the number of tracks in the stitch-unfriendly region τ, and zero otherwise. Based on these parameters, Equation 3.22 defines the cost of moving to a neighbor v_s from the current vertex v_r.

$$\text{cost}(v_s) = A_{rs}^{\text{stitch}} + A_s^{\text{stitch}} - \rho(v_r, v_s) \tag{3.22}$$

During the routing process, our algorithm records both the detour number and the cost for each explored vertex v_r. Then, neighbors v_s are explored in non-increasing order of their computed cost, thus prioritizing paths with higher routing resources and lower stitch penalties.

Following the exploration phase, the *Retrack phase* begins, wherein the router backtracks from the target to the source to find the final route. This phase aims to reduce the number of bends and to select a path with maximum available routing capacity. At each step, it considers a triplet of vertices: the current vertex v_1, its neighbor v_2, and v_2's neighbor v_3. Initially, v_1 is set to the target vertex t, and the detour numbers are arranged such that $D[v_1] \geq D[v_2] \geq D[v_3]$. If either v_2 or v_3 is the source, the retracking process terminates and the full path is returned.

As illustrated in Figure 3.8, there are twelve possible spatial patterns in which the triplet v_1, v_2, v_3 may appear in the graph G. However, not all patterns are valid in every case—some neighbors may not have been visited during the exploration phase and are thus skipped. Among the valid patterns, those involving bends are discarded if the available stitch capacity A_r^{stitch} at the bend is zero. For each remaining candidate pattern, the *bottleneck cost* is computed as $\min(\text{cost}(v_1), \text{cost}(v_2), \text{cost}(v_3))$. If the pattern includes a bend, a bend penalty is subtracted from $\text{cost}(v_2)$. If the bend occurs at a stitch-holding cell, a stitch penalty is also deducted. The pattern with the maximum bottleneck cost is selected, and the process continues by assigning v_3 as the new current vertex v_1, repeating this retrack procedure until the source vertex is reached. Algorithm 3.2 summarizes the entire method.

To demonstrate the capability of SCMR, consider a 6×6 routing grid where the source is at cell $(0, 0)$ and the target is at cell $(5, 5)$. This example models a previously unrouted two-pin flat net, which is often encountered after solving the ILP for the k^{th} box. In the grid, cells $(2, l)$, for all $l = 0$ to 5, are designated as stitch-holding cells containing a stitch-line and a stitch-unfriendly region containing two adjacent tracks of the stitch-line. Additionally, routing resources vary across the grid; some edges offer high capacity, while others are more constrained.

The exploration phase of SCMR begins at the source cell $(0, 0)$ with a detour number of zero. Neighboring vertices such as $(0, 1)$ and $(1, 0)$ are evaluated based on their detour numbers and routing capacities. Suppose both neighbors have edge stitch capacities of 1, vertex stitch capacities of 1, and are not stitch-holding. In this case, the cost becomes:

$$\text{cost}(v_s) = \text{cost}(v_r) = A_{rs}^{\text{stitch}} + A_s^{\text{stitch}} - \rho(v_r, v_s) = 1 + 1 - 0 = 2$$

These vertices are prioritized and added to the routing queue. As the exploration progresses toward the target, the router eventually considers paths through stitch-holding cells like $v_r = (2, 0)$ and $v_s = (2, 1)$. For instance, moving from v_r to v_s with edge and vertex capacities of 1 each incurs a stitch penalty $\rho(v_r, v_s) = 3$ due to the stitch-unfriendly region. In this case, the cost becomes:

$$\text{cost}(v_s) = \text{cost}(v_r) = A_{rs}^{\text{stitch}} + A_s^{\text{stitch}} - \rho(v_r, v_s) = 1 + 1 - 3 = -1$$

Algorithm 3.2: Stitch-Constrained Maze Routing (SCMR)

Input : Grid graph $G = (V, E)$, Source vertex $s \in V$, Target vertex $t \in V$, Stitch-holding cells, stitch-unfriendly regions τ, Edge stitch capacities A_{rs}^{stitch}, Vertex stitch capacities A_s^{stitch}

Output: Routing path P from s to t

Initialize: detour number $D[v] \leftarrow \infty$, cost $C[v] \leftarrow -\infty \ \forall v \in V$;
$D[s] \leftarrow 0$, $C[s] \leftarrow 0$ and Open list $Q \leftarrow \{s\}$;
`// Exploration Phase:`
while *Q is not empty* **do**
    Pop v_r from Q with highest $C[v_r]$;
    foreach *neighbor v_s of v_r* **do**
        Compute detour number $D[v_s]$ from v_r based on Hadlock's rule;
        `// Compute stitch penalty` $\rho(v_r, v_s)$
        if *v_r and v_s are stitch-holding and in τ* **then**
            $\rho(v_r, v_s) \leftarrow$ number of tracks in τ;
        else
            $\rho(v_r, v_s) \leftarrow 0$;
        Compute cost: $C[v_s] \leftarrow A_{rs}^{\text{stitch}} + A_s^{\text{stitch}} - \rho(v_r, v_s)$;
        if *$C[v_s]$ improves previous value* **then**
            Update parent of v_s to v_r, add v_s to Q;
`// Retrack Phase:`
$P \leftarrow [\,]$, $v_1 \leftarrow t$;
while $v_1 \neq s$ **do**
    Let v_2 be the parent of v_1, and v_3 be the parent of v_2;
    If $v_2 = s$ or $v_3 = s$ then Append remaining path to P and **break**;
    Identify all valid triplets (v_1, v_2, v_3) among explored nodes;
    Discard patterns with bends if $A_{v_2}^{\text{stitch}} = 0$;
    For each valid triplet, compute:
        Bottleneck cost $B = \min(C[v_1], C[v_2], C[v_3])$;
        If bend at v_2, subtract bend penalty from $C[v_2]$;
        If v_2 is stitch-holding, subtract stitch penalty;
    Select pattern with maximum bottleneck cost;
    Append v_1 to P, set $v_1 \leftarrow v_3$;
return *P as the routed path from s to t*;

The small value of $\text{cost}(v_s)$ and $\text{cost}(v_r)$ makes this path less desirable. In contrast, alternate routes that avoid stitch-holding paths, such as through $(2, 0)$ and $(3, 0)$ with capacities 1 and zero penalty, yield higher costs and are thus prioritized.

$$\text{cost}(v_s) = \text{cost}(v_r) = A_{rs}^{\text{stitch}} + A_s^{\text{stitch}} - \rho(v_r, v_s) = 1 + 1 - 0 = 2$$

Once the target is reached, the retrack phase begins. Starting from the target cell $(5, 5)$, the algorithm considers valid triplets (v_A, v_B, v_C), and we have four possible valid triplets here:

- $\{(5, 5), (4, 5), (3, 5)\}$
- $\{(5, 5), (4, 5), (4, 4)\}$
- $\{(5, 5), (5, 4), (4, 4)\}$
- $\{(5, 5), (5, 4), (5, 3)\}$

Each pattern is evaluated based on whether it was visited during exploration, whether a bend occurs, and the stitch capacity at the bend point. If a bend is detected at a stitch-holding cell with zero stitch capacity, the pattern is discarded. For valid patterns, the bottleneck cost is computed as the minimum of the costs of the three vertices. If a bend is involved in a triplet like $\{(5, 5), (4, 5), (4, 4)\}$ or $\{(5, 5), (5, 4), (4, 4)\}$, a bend penalty is subtracted from $\text{cost}(v_B)$, and a stitch penalty is subtracted if the bend occurs at a stitch-holding cell. The pattern with the highest bottleneck cost is selected, and the retrack process continues by updating $v_A \leftarrow v_C$ until the source is reached.

This example demonstrates how SCMR balances multiple routing constraints. The detour number ensures minimal deviation from the shortest path, while the stitch penalty discourages routing through stitch-unfriendly regions. The routing and vertex capacities guide the algorithm toward paths with higher available resources. The retrack phase further refines the route by avoiding bends and selecting high-quality paths based on bottleneck costs. This results in more reliable and efficient routing compared to traditional Hadlock-based methods.

3.7 Evaluation of our Stitch-avoiding Global Router

Our proposed stitch-avoiding global router is implemented in Java and executed on a Linux workstation equipped with a 2.40 GHz Intel® Xeon® CPU and 32 GB of RAM. We employ FLUTE [CW07] for RSMT construction and use GLPK [Fre12] as the ILP solver. For evaluation, we consider the Faraday suite as shown in Table 3.4 from the ICCAD'04 Mixed-Size Placement Benchmarks [oCADI21, ACR+04].

Table 3.4: Faraday benchmark circuits of ICCAD04 [oCADI21]

Circuits	#Layers	Size(μm^2)	#Nets	#Pins
DMA	6	408.4 × 408.4	13256	73982
DSP1	6	706 × 706	28447	144872
DSP2	6	642.8 × 642.8	28431	144703
RISC1	6	1003.6 × 1003.6	34034	196677
RISC2	6	959.6 × 959.6	34034	196670

Placement is performed using NTUplace3 [CJH+08], and the output is subsequently used as input to our global router. The spacing between two stitch-lines is defined as 15× the routing pitch.

Table 3.5 presents the number of Steiner branches and Steiner points before and after applying Steiner Branch Shifting (SBS) and Steiner Point Shifting (SPS). The number of Steiner branches and points located on stitch-lines are denoted by $\#SB^{SL}$ and $\#SP^{SL}$, respectively, while $\#SB^{\tau}$ and $\#SP^{\tau}$ represent the corresponding counts within stitch-unfriendly regions. The half-perimeter wirelength is reported as $HPWL$. Applying SBS leads to a reduction in $\#SB^{SL}$, $\#SP^{SL}$, $\#SB^{\tau}$, and $\#SP^{\tau}$ by approximately 70%, 59%, 44%, and 31%, respectively, at the cost of a negligible 0.14% increase in HPWL. On the other hand, SPS achieves complete elimination (100% reduction) of $\#SB^{SL}$ and $\#SP^{SL}$, along with 35% and 25% reductions in $\#SB^{\tau}$ and $\#SP^{\tau}$, respectively, incurring only a 0.07% overhead in HPWL.

Table 3.6 compares the performance of our proposed global router against two variants of the traditional *BoxRouter* [CP06]: the standard version and a modified version with reduced routing capacities assigned to stitch-holding cells. Due to the unavailability of BoxRouter's source code, we re-implemented it in Java. In the

Table 3.5: Number of Steiner Branches (SB) and Steiner Points (SP) before and after performing SBS and SPS

	Metrics	Circuits					Ratio
		DMA	DSP1	DSP2	RISC1	RISC2	
Initial: Before SBS and SPS	$\#SB^{SL}$	1522	2184	2070	3854	3949	1.00
	$\#SP^{SL}$	1240	1696	1591	3091	3183	1.00
	$\#SB^{\tau}$	3904	6157	6295	10082	10108	1.00
	$\#SP^{\tau}$	3207	4746	4803	8075	8105	1.00
	HPWL(μm)	544183.62	1143573.09	1034884.39	1669906.97	1688466.52	1.00
After SBS	$\#SB^{SL}$	494	610	552	1195	1201	0.2984
	$\#SP^{SL}$	525	654	597	1288	1307	0.4047
	$\#SB^{\tau}$	2307	3258	3450	5702	5759	0.5605
	$\#SP^{\tau}$	2327	3165	3255	5586	5709	0.6926
	HPWL(μm)	545049.31	1145058.71	1036381.16	1672344.89	1690820.61	1.0014
	CPU(ms)	19	24	29	41	44	-
After SBS and SPS	$\#SB^{SL}$	0	0	0	0	0	0.00
	$\#SP^{SL}$	0	0	0	0	0	0.00
	$\#SB^{\tau}$	1500	2103	2366	3712	3732	0.6550
	$\#SP^{\tau}$	1769	2357	2468	4216	4312	0.7545
	HPWL(μm)	545577.02	1145740.70	1037014.39	1673591.58	1692085.39	1.0007
	CPU(ms)	14	26	23	47	41	-

table, $\#TVOF$ denotes the total number of vertex capacity overflows, and $\#MVOF$ indicates the maximum vertex overflow observed in any G-cell. The total wirelength of global routing paths is represented by WL. In our implementation, we set the parameter $\varphi = 1$ in Equation 3.6 of the stitch-constrained ILP, and apply a bend penalty of 1 in the maze routing phase.

As shown in Table 3.6, our proposed global router successfully eliminates all stitch-line Steiner branches and points, achieving a 100% reduction in both $\#SB^{SL}$ and $\#SP^{SL}$. Although SBS and SPS slightly increase the density of Steiner points near stitch-holding cells, the resulting $TVOF$ and $MVOF$ values confirm that vertex capacities were not exceeded. Notably, our router avoids any overflow for the RISC2

Table 3.6: Comparison of Our Proposed Stitch-avoiding Global Router with Traditional Global Router

	Parameters	Circuits					Ratio
		DMA	DSP1	DSP2	RISC1	RISC2	
BoxRouter [CP06]	$\#SB^{SL}$	1522	2184	2070	3854	3949	1.00
	$\#SP^{SL}$	1240	1696	1591	3091	3183	1.00
	TVOF	3904	0	0	0	1	1.00
	MVOF	3207	0	0	0	4	1.00
	WL (μm)	556443.08	1163974.86	1057605.85	1705038.18	1720034.83	1.00
	CPU(s)	20.52	45.72	45.45	81.83	77.29	1.00
BoxRouter [CP06] with Reduced Capacities	$\#SB^{SL}$	1522	2184	2070	3854	3949	1.00
	$\#SP^{SL}$	1240	1696	1591	3091	3183	1.00
	TVOF	2307	0	0	0	0	1.00
	MVOF	2327	0	0	0	0	1.00
	WL (μm)	556443.08	1163974.86	1036381.16	1705102.18	1720050.82	1.00
	CPU (s)	20.52	49.19	46.85	90.68	87.54	1.1005
Our Proposed Stitch-avoiding Global Router	$\#SB^{SL}$	0	0	0	0	0	0.00
	$\#SP^{SL}$	0	0	0	0	0	0.00
	TVOF	0	0	0	0	0	0.0
	MVOF	0	0	0	0	0	0.0
	WL (μm)	558491.43	1166902.24	1060317.83	1709656.30	1724911.15	1.0028
	CPU (s)	25.79	53.19	53.20	91.80	92.39	1.1682

circuit, unlike *BoxRouter*. Overall, our stitch-avoiding router achieves 100% routability for all nets, with a marginal 0.28% increase in wirelength and a 16% increase in runtime.

To evaluate the impact of our global router on stitch-line-induced violations, we perform detailed routing using the traditional *RegularRoute* tool [ZC12]. To prevent vertical routing violations, vertical routing is blocked on tracks aligned with stitch-lines. As the *RegularRoute*'s source code is also unavailable, we have implemented it in Java. Table 3.7 compares the performance of our proposed stitch-avoiding global router (followed by *RegularRoute*) with two baseline configurations: *Baseline I*, which

Table 3.7: Effectiveness of the proposed Stitch-avoiding Global Router after detailed routing

	Parameters	Circuits					Ratio
		DMA	DSP1	DSP2	RISC1	RISC2	
Baseline Router I: BoxRouter [CP06] and Regular-Route [ZC12]	$\#VV$	3589	7469	7410	10799	3949	1.00
	$\#SHP$	1626	3496	3462	5246	3183	1.00
	$\#VRV$	0	0	0	0	0	-
	$\%Rout.$	95.94	96.56	96.69	95.92	96.09	1.00
	CPU(s)	421	1419	1296	3016	2944	1.00
Baseline Router II: BoxRouter [CP06] with Reduced Capacities and Regular-Route [ZC12]	$\#VV$	3587	7453	7408	10798	10939	1.02
	$\#SHP$	1629	3502	3480	5264	5537	1.0008
	$\#VRV$	0	0	0	0	0	-
	$\%Rout.$	95.85	96.59	96.71	95.95	96.18	1.004
	CPU(s)	406	1418	1304	3039	2986	1.1006
Our Proposed Stitch-avoiding Global Router and Regular-Route [ZC12]	$\#VV$	2990	6225	6227	8778	8891	0.8236
	$\#SHP$	1513	3246	3220	4993	5218	0.9378
	$\#VRV$	0	0	0	0	0	-
	$\%Rout.$	96.90	97.11	97.74	96.09	96.48	1.006
	CPU(s)	406	1413	1303	3035	2955	1.0017

uses *BoxRouter* followed by *RegularRoute*, and *Baseline II*, which uses *BoxRouter* with reduced routing capacities for stitch-holding cells, also followed by *RegularRoute*.

The metrics reported include the number of via violations ($\#VV$), short polygons ($\#SHP$), vertical routing violations ($\#VRV$), and overall routability percentage ($\%Rout.$). Due to the explicit blocking of stitch-line-aligned vertical tracks, $\#VRV$ is zero across all router configurations.

Table 3.7 shows that Baseline Router II does not perform as well as Baseline Router I in reducing stitch-line violations, indicating that merely reducing the routing capacities of stitch-holding cells is insufficient. In contrast, our stitch-avoiding global

router, when followed by *RegularRoute*, achieves significant improvements—reducing $\#VV$ and $\#SHP$ by an average of 17.3% and 6.21%, respectively, compared to Baseline I. Furthermore, our approach results in a 1.89% reduction in total routed wirelength and a 0.6% improvement in routability.

It is worth noting that the placement tools used in prior works, such as [LFC15] and [PBSK17] for the Faraday benchmark suite, were not disclosed. Since placement significantly affects the number of stitch-line violations, a direct comparison with these works is not feasible without this information.

The evaluation was performed using established academic routing frameworks, namely *BoxRouter* [CP06] for global routing and *RegularRoute* [ZC12] for detailed routing, implemented consistently within the same experimental environment to ensure a fair comparison. Direct comparison with commercial industrial routing tools was not feasible due to the limited accessibility of proprietary EDA toolchains, restricted access to industrial routing infrastructures, and the lack of publicly available commercial stitch-aware routing implementations for MEBL environments.

Accordingly, the proposed methodology was evaluated on the widely adopted ICCAD'04 Faraday benchmark circuits, which comprise large-scale industrial-style routing instances. The experimental results demonstrate that the proposed stitch-aware routing framework consistently improves manufacturability metrics over traditional routing approaches while maintaining competitive runtime and routability characteristics.

3.7.1 Discussion on the Optimality and Scalability of the Proposed Routing Framework

The proposed routing framework is designed to obtain high-quality manufacturability-aware routing solutions while maintaining computational scalability, rather than pursuing strict global optimality for large-scale routing instances. The stitch-avoiding global routing problem simultaneously incorporates routing-capacity constraints, stitch-avoiding capacities, vertex capacities, congestion considerations, and manufacturability constraints, resulting in a highly constrained combinatorial optimization problem. The stitch-constrained ILP formulation optimizes routing quality by consider-

ing congestion-aware weights, stitch-aware priorities, path flexibility, and wirelength simultaneously.

For smaller routing regions, the ILP solver computes the optimal solution for the formulated optimization problem within the generated candidate path space. However, the overall routing framework does not guarantee global optimality over the complete routing search space because the optimization is performed on a finite set of candidate paths generated during the preprocessing stage. Furthermore, as the routing problem size increases, obtaining globally optimal solutions becomes computationally intractable due to the NP-hard nature of routing optimization. Consequently, the proposed framework emphasizes practical routing quality, reduced stitch-line violations, improved routability, and computational scalability while preserving the flexibility required for large-scale routing applications. The effectiveness of these design choices is validated through the experimental results presented in Section 3.7.

The scalability of the proposed framework is enhanced through decomposition of multi-pin nets into two-pin segments, stitch-avoiding Steiner shifting, priority-based routing of less flexible nets, and stitch-constrained maze routing for unresolved segments. Consequently, although the underlying optimization problem remains NP-hard, the progressive box-expansion framework effectively controls the optimization problem size by localizing routing decisions, optimizing only subsets of nets simultaneously, and deferring difficult routing cases to stitch-constrained maze routing rather than excessively enlarging the ILP. Furthermore, the framework combines heuristic Steiner shifting, localized ILP optimization, and maze-routing-based refinement to achieve a balance between routing quality and computational tractability for large routing instances.

The scalability of the proposed framework is demonstrated through experimental evaluation on the Faraday benchmark circuits, as shown in Table 3.4, circuits containing up to 34,034 nets and 196,677 pins. The benchmark suite ranges from DMA with 13,256 nets and 73,982 pins to RISC2 with 34,034 nets and 196,677 pins. Despite the substantial increase in design complexity, the proposed decomposition strategy enables practical runtime performance while maintaining routing quality, achieving 100% routability during global routing, preserving zero stitch-line Steiner violations after routing, and significantly reducing stitch-line violations.

3.7.2 Limitations and Correctness of the Proposed Routing Framework

Although the proposed routing framework demonstrates significant improvements in manufacturability-aware routing performance, certain routing scenarios remain inherently challenging due to physical design constraints. In particular, highly congested routing regions with limited routing resources and dense stitch-line distributions may restrict the ability of the framework to completely eliminate stitch-sensitive routing patterns. This limitation is reflected in the Steiner Branch Shifting (SBS) and Steiner Point Shifting (SPS) stages. When sufficient routing space outside stitch-unfriendly regions is unavailable because of routing-capacity constraints, further relocation of Steiner branches or Steiner points may not be feasible without violating routing constraints.

Similarly, routing instances classified as Category 2 and Category 3 within the proposed box-expansion framework represent more challenging routing scenarios in which simple L-shaped routing patterns are insufficient. Such nets require additional routing bends, more flexible maze-routing exploration, and, in some cases, iterative rerouting before feasible manufacturability-aware solutions can be obtained.

Despite these limitations, the proposed framework guarantees routing correctness by enforcing all routing constraints throughout the optimization process. In particular, stitch-avoiding edge-capacity and vertex-capacity constraints ensure that the utilization of routing resources never exceeds their allowable capacities, i.e., $U_{rs} \leq C_{rs}^{\text{Stitch}}$ and $U_r \leq C_r^{\text{Stitch}}$.

Furthermore, the ILP constraints presented in Equation 3.7 to Equation 3.12 explicitly enforce routing legality by satisfying edge-capacity and vertex-capacity constraints throughout the optimization process. Consequently, although the proposed framework does not guarantee globally optimal routing solutions for all routing instances, it consistently preserves routing correctness, manufacturability-aware legality, and constraint satisfaction.

3.8 Concluding Remarks

This work introduces a global routing technique for Multiple Electron Beam Lithography (MEBL) that actively prevents stitch-line violations. The proposed approach integrates dedicated net-shifting strategies with integer linear programming (ILP), that is composed of several constraints to avoid stitch-line violations and maze routing with stitch-line penalties and other multiple new components to minimize stitch-line violations during global routing. By explicitly accounting for stitch constraints during global routing, our method significantly reduces the count of stitch-line violations compared to conventional global routing techniques.

Experimental evaluations on benchmark circuits demonstrate substantial improvements in stitch-line compliance without compromising other key routing metrics such as half-perimeter wirelength (HPWL) and CPU runtime. These results highlight the effectiveness of incorporating stitch-avoidance mechanisms at the global routing stage.

While the current chapter focuses on stitch-aware global routing, further performance gains may be achieved by coupling our proposed approach with popular commercial and academic detailed routers. In particular, the design of a stitch-avoiding detailed router has the potential to further suppress stitch-line violations by addressing lithography constraints at a finer granularity. Therefore, to address these challenges, the next chapter presents an efficient, high-performance stitch-avoiding detailed routing methodology for MEBL.

CHAPTER

FOUR

STITCH-AVOIDING DETAILED ROUTING FOR MULTIPLE E-BEAM LITHOGRAPHY

Contents

4.1 Introduction **103**

4.2 Motivation **103**

4.3 Preliminaries and Problem Formulation **104**

4.3.1 Preliminaries 104

4.3.2 Problem Formulation 107

4.3.3 Net Classification 108

4.4 Overview of Our Proposed Detailed Routing Framework for MEBL **110**

4.5 Detailed routing for Single-Cell Confined Nets (SCNs) with stitch avoidance **113**

4.6 Stitch-avoiding Detailed Routing of MCNs **114**

4.6.1 Routing Vertical Segments of MCNs 115

4.6.2 Routing of horizontal segments in MCMPNs 126

4.6.3 Routing of MCSPNs having all HSegments in one HPanel 128

4.6.4 Discussion on the Computational Overhead of the Proposed Framework . . . 130
4.7 Experimental Results . . . 131
4.8 Conclusion . . . 135

4.1 Introduction

In this chapter, we introduce a comprehensive full-routing methodology aimed at further decreasing stitch-related violations by incorporating a *stitch-avoiding detailed router*. While the *stitch-avoiding global router* provides the foundational routing for all nets, an additional dedicated detailed routing stage is required to specifically address stitch-induced violations. Using the 2D global routing outcome generated by our stitch-avoiding global routing approach, the detailed router performs stitch-avoiding routing to reduce issues such as via violations, vertical routing conflicts, and short polygon errors. We evaluate the proposed routing framework on the Faraday [oCADI21] benchmark circuits, demonstrating a substantial improvement in violation reduction.

The organization of this chapter is as follows: Section 4.2 explains the motivation behind designing a stitch-avoiding detailed router. Section 4.3 introduces key definitions, formulates the problem, and summarizes the overall framework. The methods developed in this work are described in detail in Section 4.5 and Section 4.6. Experimental findings are reported in Section 4.7, and Section 4.8 presents concluding remarks along with possible directions for future research.

4.2 Motivation

The DP-based placement method in [LYZ+17] improves routability by refining local congestion, but its scope is limited as it primarily addresses violations from fixed pin locations and does not adequately handle stitch-related issues in later routing stages. The method in [LFC15] augments a conventional global router with an addi-

tional congestion constraint, which reduces short-polygon violations mainly in track assignment as well as in detailed routing. However, it simultaneously increases via violations, creating an unfavorable trade-off between violation types.

The perturbation-based post-routing approach in [PBSK17] mitigates stitch-line violations through localized adjustments, yet introduces a large number of vertical routing violations, making it unsuitable for congested designs. Similarly, the smart-boundary technique in [HF20] provides finer stitch control by adaptively defining routing boundaries, but its reliance on complex e-beam control limits scalability and practical deployment.

More recently, a stitch-avoiding global routing framework for MEBL was proposed in [SPBSK22], which explicitly incorporates stitch-line constraints into global routing and achieves a substantial reduction in stitch-line violations. However, as this method operates only at the global routing stage, residual violations often persist into detailed routing, reducing its overall effectiveness.

In summary, while prior work has advanced violation reduction at various stages of the design flow, no existing method delivers a comprehensive solution at the detailed routing level. A robust MEBL detailed router must therefore integrate stitch-line constraints while simultaneously minimizing via as well as short-polygon violations and eliminating vertical routing violations. Such an approach would overcome the limitations of placement, global routing, and post-routing techniques, offering a more practical and manufacturable solution for advanced MEBL circuit layouts.

4.3 Preliminaries and Problem Formulation

This section provides the necessary prerequisites and formally defines the problem.

4.3.1 Preliminaries

Definition 4.3.1. In the grid-graph representation, the layout is divided into sectors known as *grid cells* (G-cells), each labeled as c_i, as illustrated in Figure 4.1(a).

Definition 4.3.2. A *horizontal panel* (HPanel) Π_m^H refers to the collection of all G-cells located in the m^{th} row of a horizontal routing layer (HLayer). The HPanels are

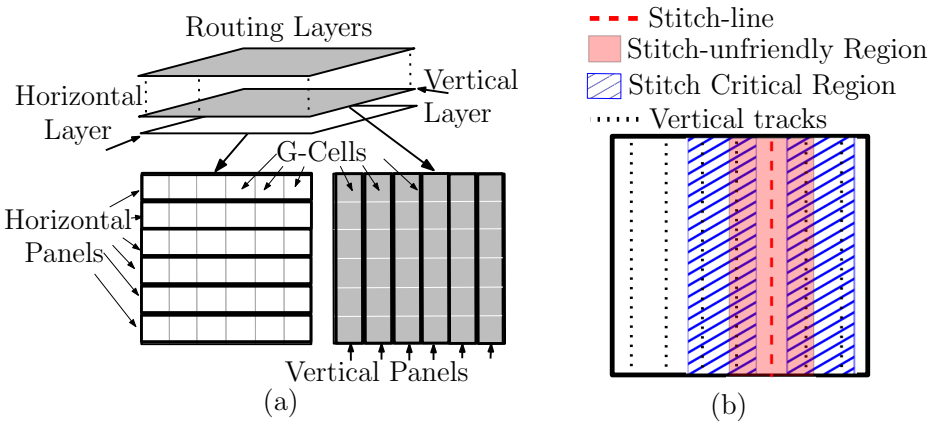


Figure 4.1: (a) Illustration of the G-cell along with the horizontal and vertical panels; (b) depiction of the Stitch Critical Region.

illustrated in Figure 4.1(a).

Definition 4.3.3. A *vertical panel* (VPanel) Π_m^V represents the group of G-cells situated in the m^{th} column of a vertical routing layer (VLayer). The VPanels are depicted in Figure 4.1(a).

Definition 4.3.4. A *Stitch Critical Region* refers to the collection of tracks located within or directly next to a stitch-unfriendly area. In Figure 4.1(b), this region is illustrated using a blue diagonal hatched pattern.

Table 4.1 gives the list of symbols used in this chapter and their meanings.

Table 4.1: List of Symbols and Their Meanings

Symbol	**Meaning**
c_i	i^{th} G-cell.
Π_m^H	Horizontal Panel consists of a set of G-cells of m^{th} row of a horizontal routing layer.
Π_m^V	Vertical Panel consists of a set of G-cells of m^{th} column of a vertical routing layer.
N	A set of nets.

Continued on next page

Table 4.1 continued from previous page

Symbol	**Meaning**
L	A set of routing layers.
S	A set of stitch-line locations.
$G(N)$	A global routing solution for each net.
$R(N)$	Routing Solutions for each net $N_i \in N$.
$V(R)$	The number of via violations.
$P(R)$	The number of short polygon violations.
$H(R)$	The number of vertical routing violations.
P_n	Pins of net n.
$P_j^{N_i}$	j^{th} pin in N_i net.
Π_i	A specific vertical panel.
VSEG	The set of vertical segments in Π_i.
Ψ_j	A specific vertical segment in set VSEG
VTracks	A set of vertical tracks.
$VTRACK_j$	Specific vertical tracks where Ψ_j can be assigned.
T_k	A track in $VTRACK_j$ corresponds to a potential assignment for Ψ_j.
(Ψ_j, T_k)	Represents a vertex of segment conflict graph (SCG).
v^*	A vertex with maximum benefit.
$W(\Psi^{N_i}, T_j)$	A weight assigned to each vertex (Ψ^{N_i}, T_j) of the SCG, where larger weight indicates a higher routing priority.
$L(\Psi^{N_i})$	The length of the vertical segment Ψ^{N_i}.
$f(\Psi^{N_i}, T_j)$	The function used to assign a higher weight to a vertex of the segment conflict graph.
$avgSegL$	The average segment length of the panel.
$\Omega(\Psi^{N_i})$	The function quantifies the average number of VSegments that cross the G-cell boundaries traversed by the segment Ψ^{N_i}.
B	The set of G-cell boundaries intersected by Ψ^{N_i}.
λ_b	Number of VSegments crossing the boundary line segment b.
$\chi(\Psi^{N_i}, T_j)$	Cost of establishing the VSegment-to-pin connection.
η_1	Count of detailed routing grid edges crossed in *preferred* routing direction.

Continued on next page

Table 4.1 continued from previous page

Symbol	Meaning
η_2	Count of detailed routing grid edges crossed in *non-preferred* routing direction.
η_3	The number of vias used in the path.
$\xi(\Psi^{N_i}, T_j)$	The stitch penalty associated with assigning the VSegment Ψ^{N_i} to the target pin T_j.
η_{SP}	The number of short polygons generated in VSegment-to-pin connection.
η_{VV}	The number of via violations arising in the VSegment-to-pin connection.
$L(\Psi^{N_i})$	The segment length of Ψ^{N_i}.
$W'(\Psi^{N_i}, T_j)$	Incorporates the number of possible via positions for horizontal connections.
$\upsilon(\Psi^{N_i}, T_j)$	The minimum number of possible via positions available at the two ends of the segment.
p	The number of panels.
$W''(\Psi^{N_i}, T_j)$	Weight assignment during constructing SCG and solving MWIS are similar to that for VSegment assignment of MCNs.
$\#Layers$	The number of layers.
$\#Nets$	The number of nets.
$\#Pins$	The number of pins.
$\#VV$	The number of via violations.
$\#SHP$	The number of short polygons.
$\#VRV$	The number of vertical routing violations.
$\%Rout.$	The routability percentage.

4.3.2 Problem Formulation

The *Stitch-Avoiding Detailed Routing Problem* for MEBL can be described as follows: Given,

- A collection of nets $N = \{N_1, N_2, \ldots, N_n\}$,
- A group of routing layers $L = \{L_1, L_2, \ldots, L_k\}$,
- A set of stitch-line positions $S = \{S_1, S_2, \ldots, S_l\}$,
- A corresponding global routing outcome $G(N)$ for every net,

Design a stitch-avoiding detailed routing technique to determine detailed routing solutions $R(N)$ for each net $N_i \in N$ that assigns net segments to routing tracks of different layers such that:

$$\begin{aligned} &\textit{minimize} && \Big(V(R) + P(R)\Big) && (4.1)\\ &\text{subject to} && R(N) \cap S = \varnothing \quad \text{(stitch-avoidance constraint)}, && (4.2)\\ &&& H(R) = 0 \quad \text{(no vertical routing violations)}, && (4.3)\\ &&& V(R) \geq 0, \quad P(R) \geq 0 && (4.4) \end{aligned}$$

Here, $V(R)$ denotes the count of via violations, $P(R)$ represents the number of short-polygon violations, and $H(R)$ indicates the vertical routing violations. Consequently, the goal is to reduce via and short-polygon issues while ensuring that no vertical routing violations occur.

4.3.3 Net Classification

Depending on how far a net extends across the G-cell grid, nets are grouped into the categories below:

- *Single-Cell Confined Net* (SCN): A net whose complete set of segments remains within one G-cell. For instance, the blue Net A in Figure 4.2 fits this class because all of its segments are contained inside G-cell 1.
- *Multi-Cell Confined Net* (MCN): A net whose segments cover more than one G-cell. This class is subdivided into:
  - *Multi-Cell Single-Panel Net* (MCSPN): A net that spans several G-cells while remaining restricted to a single HPanel or VPanel. As an example, the green Net B in Figure 4.2 qualifies as an MCSPN since all its segments lie within a single vertical panel consisting of G-cells 2, 5, and 8.
  - *Multi-Cell Multi-Panel Net* (MCMPN): A net whose segments traverse multiple G-cells and extend into more than one horizontal or vertical panel. The purple Net C in Figure 4.2 belongs to this category because it stretches across several panels in both directions.

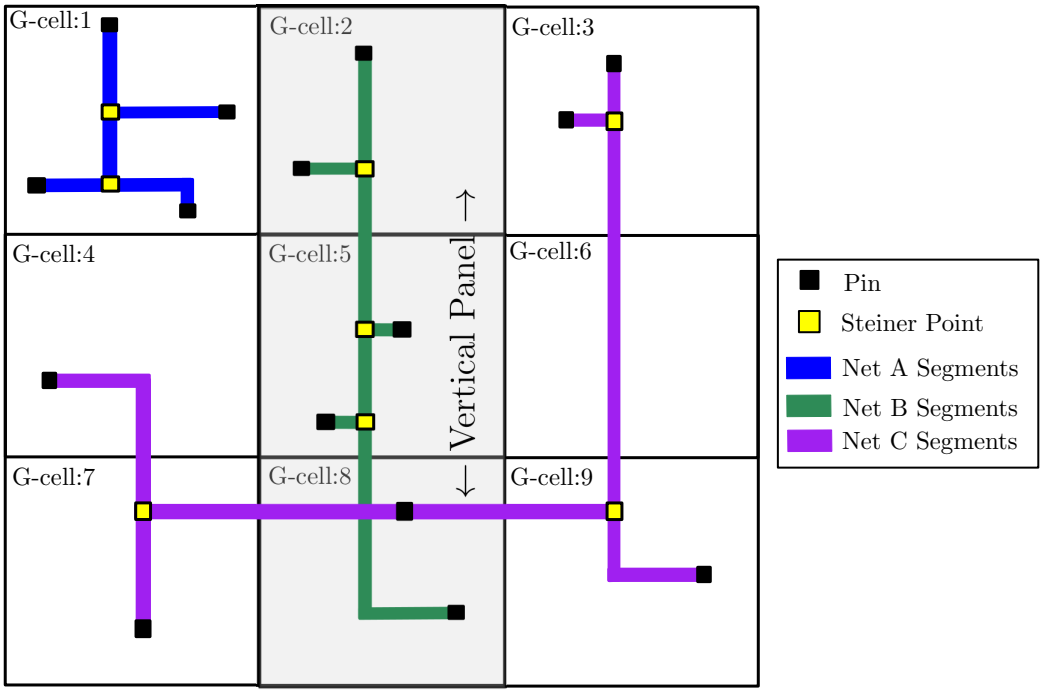


Figure 4.2: Various net types are illustrated: Net A (blue) corresponds to a Single-Cell Confined Net (SCN), Net B (green) corresponds to a Multi-Cell Single-Panel Net (MCSPN), and Net C (purple) corresponds to a Multi-Cell Multi-Panel Net (MCMPN).

4.3.3.1 Impact of the Net Classification Strategy

The proposed net-classification strategy plays a central role in the stitch-avoiding detailed routing framework by enabling routing decisions that are tailored to the routing characteristics and flexibility of different classes of nets. Rather than applying a uniform routing strategy, the classification allows the framework to select routing techniques that are more suitable for each routing scenario, thereby improving routing quality while maintaining computational efficiency.

Nets with limited routing flexibility are routed first to minimize contention for stitch-sensitive routing resources during subsequent routing stages. The proposed Single Trunk V-tree routing strategy further reduces the utilization of vertical routing tracks, which is particularly beneficial because the available vertical routing capacity

is limited near stitch-line regions.

For multi-cell routing instances, the decomposition into independent vertical and horizontal routing segments enables localized conflict management, stitch-aware track assignment, reduced routing congestion, and improved control over via generation. Consequently, routing decisions become more localized and computationally efficient while preserving routing quality.

The use of partial segment assignment preserves routing flexibility when complete segment-to-pin connections cannot be established immediately. Similarly, delayed segment routing prioritizes routing segments outside stitch-critical regions while postponing more difficult routing cases for specialized maze-routing refinement. This strategy enables congested routing regions to be handled more effectively without unnecessarily restricting feasible routing solutions.

Overall, the proposed net-classification strategy improves routing quality by enabling routing algorithms that are specifically tailored to the routing characteristics of different net classes. By prioritizing less flexible routing instances, preserving routing flexibility for complex routing scenarios, and utilizing stitch-sensitive routing resources more judiciously, the proposed framework achieves improved manufacturability-aware routing while maintaining practical routing efficiency. The effectiveness of the proposed strategy is validated experimentally in Section 4.7.

4.4 Overview of Our Proposed Detailed Routing Framework for MEBL

The stitch-avoiding detailed router proposed in this work functions directly on the 2D global routing output, eliminating the need for a separate layer-assignment step. Rather than assigning layers beforehand, the router determines them dynamically during detailed routing. This approach draws inspiration from the RegularRoute detailed router [ZC12].

RegularRoute [ZC12] routes nets with minimal bends and attempts to keep each net segment on a single track, subject to track capacity. While efficient for conventional lithography, it was not designed for MEBL and therefore cannot handle

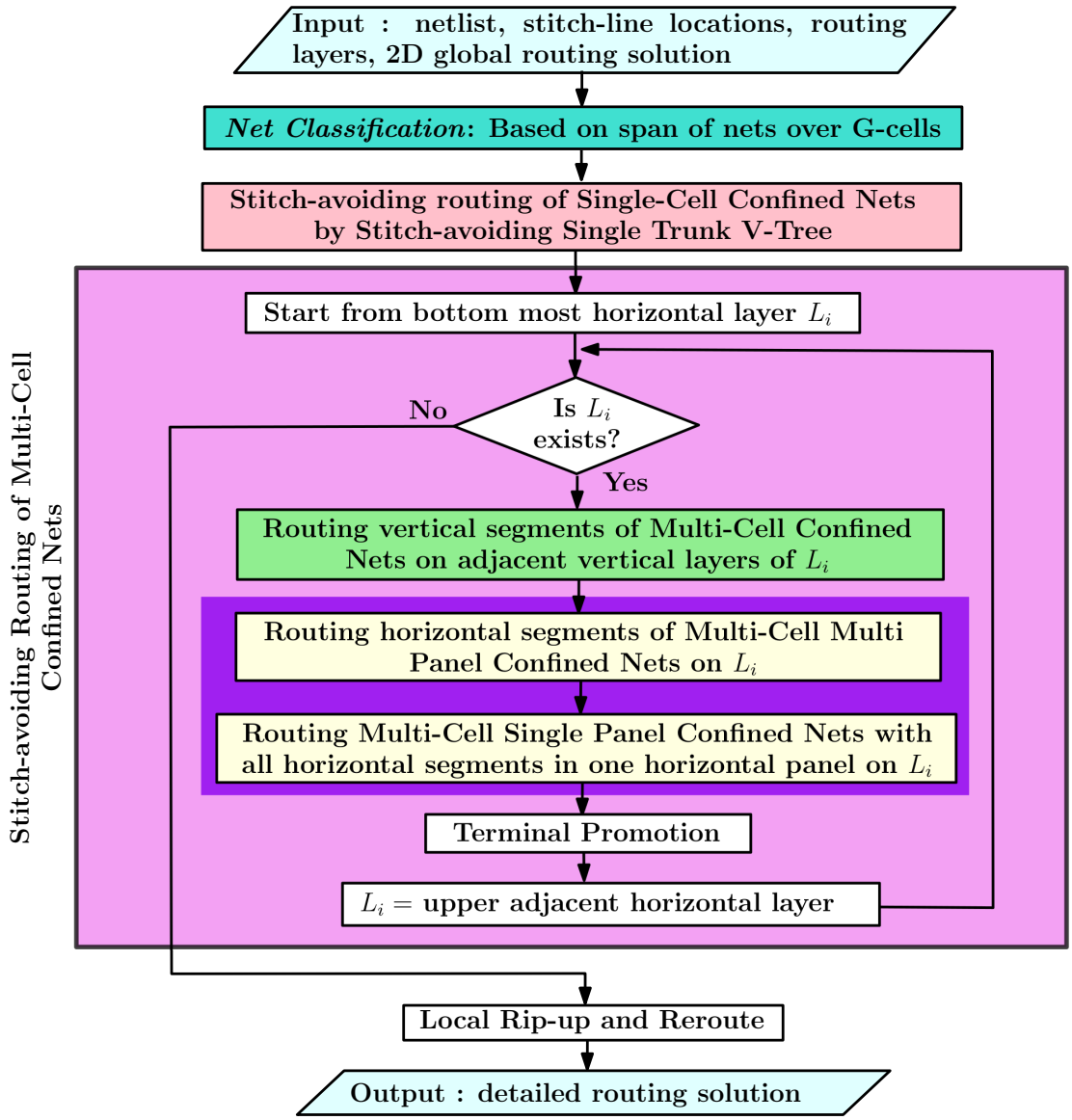


Figure 4.3: Overview of the proposed stitch-avoiding detailed routing framework

stitch-lines. Nevertheless, its segment-assignment principle is useful in MEBL, particularly under conditions of reduced vertical capacity caused by stitch-lines.

Figure 4.3 presents the complete workflow of the stitch-avoiding detailed routing methodology. Vertical tracks (VTracks) that coincide with stitch-lines are disabled to eliminate stitch-related violations. Routing begins with the Single-Cell Confined Nets (SCNs), as their limited span offers fewer routing alternatives. Because the stitch-lines run vertically, each Multi-Cell Confined Net (MCN) is divided into horizontal segments (HSegments) and vertical segments (VSegments). In contrast to RegularRoute, the proposed router handles VSegments and HSegments independently, following the order depicted in Figure 4.3.

The proposed framework performs routing and layer assignment concurrently rather than using a separate intermediate layer-assignment phase. The routing methodology adopts a bottom-up layer assignment strategy in which routing begins from the lowest routing layers, unrouted terminals are progressively promoted to upper layers, and adjacent layers are used as patch layers only when congestion prevents successful routing in the current layer.

This strategy was selected because it minimizes unnecessary via generation, non-preferred routing, and routing inside stitch-critical regions. The proposed framework additionally imposes stitch-aware restrictions during dynamic layer assignment. In particular:

- vertical tracks coinciding with stitch-lines are blocked,
- layer changes inside stitch-unfriendly regions are restricted,
- and patch layers are invoked only when required by congestion.

These constraints significantly reduce via violations and short polygon generation. The routing framework further incorporates partial segment assignment, delayed segment routing, and stitch-aware maze routing, which collectively provide adaptive layer utilization depending on local congestion and stitch-line constraints. Alternative layer-assignment strategies such as fully global layer optimization, reinforcement-learning-based layer assignment, or congestion-prediction-driven adaptive layer allocation may also be explored in future extensions. However, the present framework prioritizes practical manufacturability-aware routing while maintaining efficient runtime and high routability.

Finally, a localized rip-up-and-reroute step is applied to reduce the number of bends. It should be emphasized that fixed pins placed on stitch-lines cannot be repositioned during detailed routing and can only be shifted during placement.

4.5 Detailed routing for Single-Cell Confined Nets (SCNs) with stitch avoidance

As the vertical routing capacity decreases, it becomes necessary to utilize fewer VTracks for routing single-cell confined Nets (SCNs). To address this, we employ a *Single-Trunk V-tree* structure, which consists of a single vertical segment, referred to as the *V-trunk*. For each non-stitch SCN, the V-trunk is assigned to the VTrack that passes through the median x-coordinate of all its pins in the lowest VLayer. The V-trunk spans vertically from the minimum to maximum y-coordinates of the pins, while horizontal segments (HSegments) are connected to it in the lowest HLayer.

When multiple SCNs reside within the same G-cell, a *shifting mechanism* is applied to ensure overlap-free assignment of their V-trunks. Since the number of SCNs and available tracks per G-cell is typically small, this shifting procedure is computationally inexpensive. However, selecting an appropriate V-trunk becomes challenging in the presence of stitch-lines, as additional constraints must be satisfied to avoid routing conflicts. Algorithm 4.1 summarizes the entire method.

As an illustration, assume that metal-1 and metal-3 serve as HLayers, while metal-2 acts as the VLayer. Figure 4.4 depicts an SCN. The routing begins by placing the vertical trunk on metal-2, followed by routing all HSegments on metal-1. This initial routing configuration generates three short polygons, as shown in Figure 4.4(a). To eliminate two of them, the vertical trunk is shifted to a neighboring track, illustrated in Figure 4.4(b). To remove the remaining short polygon near pin $P_5^{N_i}$, a new via is placed outside the stitch-unfriendly region, and the HSegment connecting $P_5^{N_i}$ to this via is routed on metal-1, as shown in Figure 4.4(c). Finally, the via is linked with the vertical trunk using metal-3, shown in Figure 4.4(d).

Let N denote the number of SCNs within a G-cell and T the number of available VTracks. The median computation for each SCN requires $\mathcal{O}(\log |d_n|)$ time, where

Algorithm 4.1: Single-Trunk V-tree Assignment for SCNs

Input: Set of SCNs within a G-cell, stitch-line locations S, available VTracks and HTracks

Output: Assigned V-trunks and HSegments for all SCNs

```
foreach SCN n ∈ G-cell do
    Compute the median x-coordinate of all pins of n;
    Place a tentative V-trunk on the VTrack closest to the median x in the
     lowest VLayer;
    Extend the V-trunk from min(y_pins) to max(y_pins);
    Connect all pins to the V-trunk using HSegments in the lowest HLayer;
foreach pair of SCNs (n_i, n_j) with overlapping V-trunks do
    Apply shifting mechanism to reassign one of the trunks to the nearest
     available VTrack;
foreach SCN n do
    if V-trunk overlaps with a stitch-line in S then
        Recompute trunk placement on an alternate feasible VTrack;
        if no feasible VTrack exists then
            Mark n as unroutable at this stage;
return Final assignment of V-trunks and HSegments for all SCNs
```

$|d_n|$ denotes the degree of net n, which is usually small and can be treated as $\mathcal{O}(1)$. Thus, initial trunk placement runs in $\mathcal{O}(N)$. The pairwise overlap resolution step in the worst case requires $\mathcal{O}(N^2)$ checks; however, since N is small in practice, this step is not computationally expensive. Checking for stitch-line conflicts requires at most $\mathcal{O}(N \cdot T)$ operations. Hence, the overall runtime of the algorithm is bounded by $\mathcal{O}(N^2 + N \cdot T)$.

4.6 Stitch-avoiding Detailed Routing of MCNs

Each MCN is first decomposed into horizontal segments (HSegments) and vertical segments (VSegments). HSegments are routed within the horizontal routing layers (HLayers), while VSegments are routed within the vertical routing layers (VLayers).

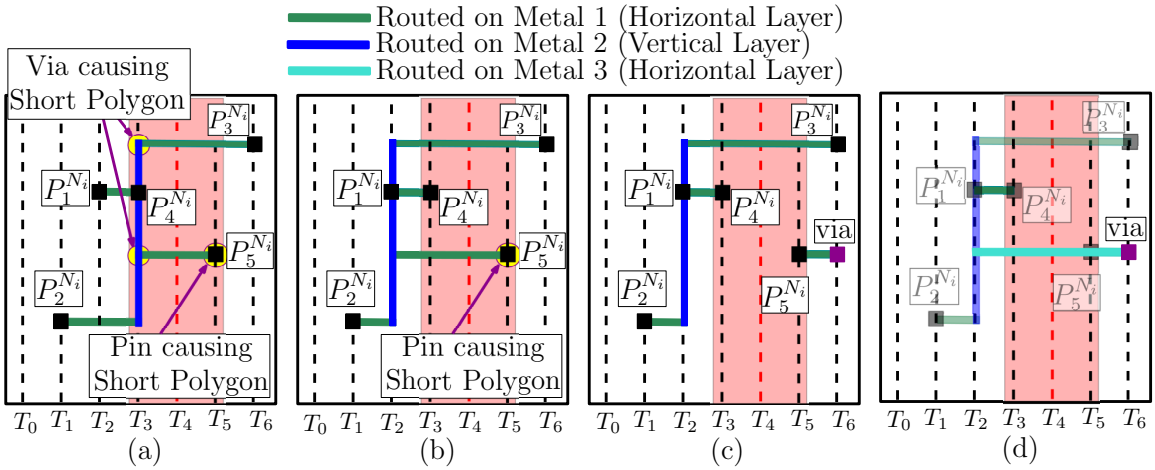


Figure 4.4: (a) A single-trunk V-tree is formed by placing a trunk at the median x-location of the pins, which results in several short polygons. (b) Shifting the vertical trunk locally removes some of these short polygons, although the violation near pin $P_5^{N_i}$ remains. (c) Adding a new HSegment along with a via eliminates the short polygon at $P_5^{N_i}$. (d) The via is then connected to the vertical trunk through a horizontal segment routed on an upper HLayer.

4.6.1 Routing Vertical Segments of MCNs

Algorithm 4.2 presents the routing workflow for handling VSegments of MCNs. Starting from the leftmost VPanel on the current VLayer, VSegments are placed onto available tracks, proceeding panel by panel. The method is organized into three primary steps:

1. Routing within the non–stitch-critical region and creating a *delayed segments list*, which stores VSegments that must be processed inside the stitch-critical area.
2. Decomposing these delayed segments and routing them within the stitch-critical region.
3. Performing partial routing of VSegments in the non–stitch-critical region.

A comprehensive explanation of these three stages is provided in Section 4.6.1.1, Section 4.6.1.2, and Section 4.6.1.3.

Algorithm 4.2: Routing of VSegments of MCNs

```
1  for each VPanel Π_i in VLayer L_i do
2      Set Delayed_Segment_List=∅;
3      if Π_i contains stitch-line then
4          Find set Track_SCR of VTracks present in Stitch Critical Region of Π_i;
5      end
6      Find set VSEG of VSegments of Π_i;
7      Construct Segment Conflict Graph (SCG) from VSEG;
8      while SCG contains vertices do
9          Find MWIS on SCG to select the best segment track pair (Ψ, T);
10         if T ∈ Track_SCR then
11             Insert (Ψ, T) in Delayed_Segment_List and reserve vertical and
                horizontal routing resource;
12         else
13             Assign segment Ψ to track T;
14             if Corresponding HSegment of Ψ is unrouted then
15                 Determine Via Position;
16             end
17             Maze routing for VSegment-to-pin Connection;
18         end
19         Update SCG ;
20     end
21     Routing of segments in Delayed_Segment_List ;
22     Partial VSegment Assignment;
23 end
```

4.6.1.1 Routing within the Non–Stitch-Critical Region and Detection of the Delayed Segments List

For every VPanel within the current VLayer, assigning VSegments is modeled as computing a Maximum Weighted Independent Set (MWIS) on a Segment Conflict Graph (SCG), similar to the method in [ZC12]. The *Segment Conflict Graph* (SCG) is defined as $SCG = (V, E)$, where V represents the vertices and E represents the edges. This graph is built sequentially, one panel at a time. Let the current vertical panel be denoted by Π_i. As described in Line 6 of Algorithm 4.2, we first compute the

set $VSEG$ of VSegments in Π_i. For each VSegment $\Psi_j \in VSEG$, we determine the set of VTracks, denoted as $VTRACK_j$, within Π_i where Ψ_j can be assigned. Each track $T_k \in VTRACK_j$ corresponds to a potential assignment for Ψ_j. We represent each such assignment as a vertex $(\Psi_j, T_k) \in V$.

For example, Figure 4.5 illustrates the construction of an SCG for a VPanel. Figure 4.5(a) illustrates four VSegments Ψ^{N_1}, Ψ^{N_2}, Ψ^{N_3}, and Ψ^{N_4} belonging to the set $VSEG$ of panel Π_k. Figure 4.5(b) depicts the current state of panel Π_k, which contains prerouted segments and a stitch-line. Since the track along the stitch-line is blocked, no segment can be routed on it. The feasible tracks for the four segments are given by:

- $\Psi^{N_1} : \{T_1, T_2, T_3, T_5\}$
- $\Psi^{N_2} : \{T_3, T_5\}$
- $\Psi^{N_3} : \{T_3, T_5, T_6\}$
- $\Psi^{N_4} : \{T_0, T_1, T_2, T_3, T_5, T_6\}$

Each feasible assignment (Ψ^{N_i}, T_j) is represented as a vertex in SCG. All vertices corresponding to different track assignments of the same segment form a clique, since exactly one track can be chosen per segment. For example, the four segments yield four distinct cliques. Edges between vertices of different segments capture conflicts: an edge exists between (Ψ^{N_p}, T_q) and (Ψ^{N_r}, T_q) if both segments select the same track T_q and overlap spatially. For instance, an edge exists between (Ψ^{N_1}, T_3) and (Ψ^{N_4}, T_3) because both select track T_3 and overlap when routed on it. Conversely, no edge exists between (Ψ^{N_3}, T_3) and (Ψ^{N_4}, T_3), as they do not overlap when both are routed on T_3.

Weights are associated with the SCG vertices. While obtaining an exact MWIS on arbitrary graphs is NP-complete, we adopt the heuristic from [ZC12], which yields a polynomial-time approximation for SCGs. Afterward, the benefit of each vertex is calculated using its assigned weight, and the vertex offering the greatest benefit is chosen for routing.

Unlike the approach in [ZC12], we introduce a revised weight formulation for assigning VSegments, explicitly accounting for stitch-line limitations. This enhancement is crucial because traditional weight models often lead to stitch-related violations in MEBL. By integrating these constraints, VSegments that are more likely to incur such violations are prioritized for routing outside the *Stitch Critical Region*. During

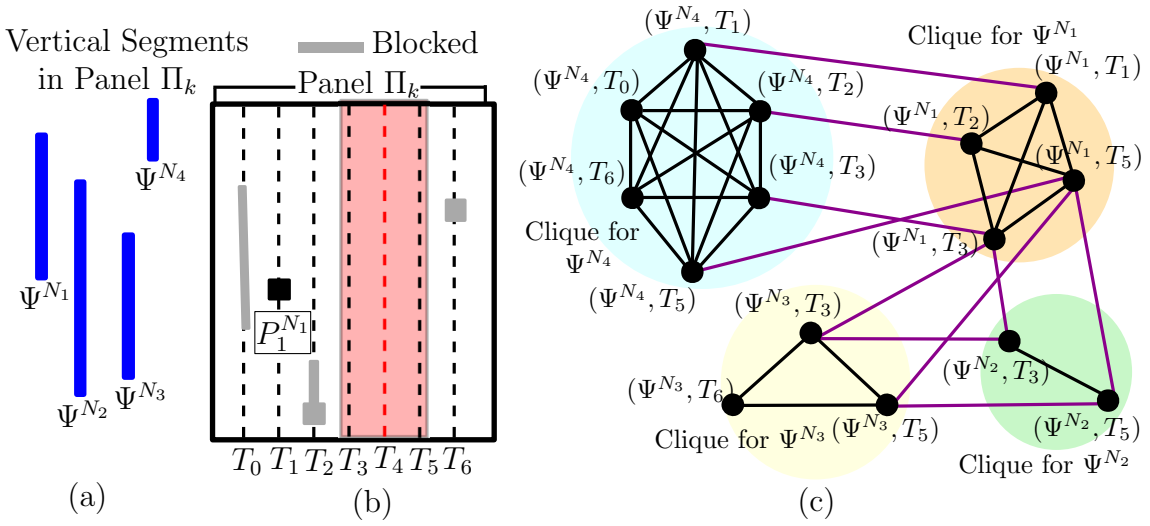


Figure 4.5: Formation of the Segment Conflict Graph (SCG); (a) VSegments located in panel Π_k; (b) Current occupancy of Π_k; (c) The resulting SCG. Black edges denote conflicts among alternative choices of a single segment, while magenta edges indicate conflicts between options belonging to distinct segments.

this process, we also determine which VSegments are forced to remain inside the *Stitch Critical Region*.

As described above, the routing decision is based on selecting the vertex v^* with maximum benefit. Depending on the location of v^*, one of the following two scenarios arises:

- **Scenario I:** If v^* corresponds to a track inside the SCR, the corresponding VSegment is *not* assigned to that track. Instead, it is labeled as a *delayed segment* and stored in the list $\mathcal{L}_{\text{delayed}}$ for deferred routing in a subsequent stage.
- **Scenario II:** If v^* corresponds to a track outside the SCR, the VSegment is immediately assigned to that track. Subsequently, maze routing is performed in the lower H-layer of the current V-layer to connect the VSegment endpoints to the corresponding pins or vias.

Next, we update the *SCG* by removing the highest-benefit vertex along with all vertices and edges adjacent to it. Subsequently, the weights of the remaining

vertices in the updated SCG are recomputed, and the MWIS problem is solved again. This iterative procedure continues until no vertices remain in SCG. Once this process is complete, we proceed to route the delayed segments stored in the set $Delayed_Segment_List$. These segments are particularly critical, as they are routed within the *Stitch Critical Region*.

We introduce the proposed weighting scheme $W(\Psi^{N_i}, T_j)$, which specifies the weight associated with each vertex (Ψ^{N_i}, T_j) in the SCG. This value reflects how strongly segment Ψ^{N_i} should be mapped to track T_j; a higher weight denotes greater routing preference. For every VPanel, the weight expression shown in Equation 4.5 is applied, integrating stitch-line restrictions to avoid stitch-related violations.

$$W(\Psi^{N_i}, T_j) = f(\Psi^{N_i}, T_j) \cdot L(\Psi^{N_i}) + \alpha_1 \cdot \Omega(\Psi^{N_i}) - \chi(\Psi^{N_i}, T_j) - \xi(\Psi^{N_i}, T_j) \quad (4.5)$$

In this context, $L(\Psi^{N_i})$ represents the length of the vertical segment Ψ^{N_i}, obtained by multiplying the count of G-cells it covers inside the VPanel with the number of HTracks contained in each G-cell. The parameter Ω captures the density of VSegments, $\chi(\Psi^{N_i})$ reflects the connection cost between the segment and its pin, and $\xi(\Psi^{N_i}, T_j)$ indicates the penalty related to stitching.

The function $f(\Psi^{N_i}, T_j)$ assigns a higher weight to a vertex of the SCG when the selected track lies within a *stitch critical region* and the corresponding segment length satisfies $L(\Psi^{N_i}) \geq avgSegL$, where $avgSegL$ denotes the average segment length of the panel. Formally, $f(\Psi^{N_i}, T_j)$ is defined in Equation 4.6 where σ is a user-specified constant and $r = \dfrac{L(\Psi^{N_i})}{avgSegL}$.

$$f(\Psi^{N_i}, T_j) = \begin{cases} (1+r)^{\sigma}, & \text{if } T_j \text{ lies within a stitch critical region and } L(\Psi^{N_i}) \geq avgSegL, \\ 1, & \text{otherwise,} \end{cases} \quad (4.6)$$

This formulation prioritizes longer VSegments in *stitch critical regions*, as routing longer segments in these regions reduces the number of VSegments present, thereby lowering the via count for horizontal connections. Furthermore, longer segments provide greater routing flexibility.

The function $\Omega(\Psi^{N_i})$ quantifies the average number of VSegments that cross the G-cell boundaries traversed by the segment Ψ^{N_i}. It is formally defined in Equation 4.7.

$$\Omega(\Psi^{N_i}) = \frac{1}{|B|} \sum_{b \in B} \lambda_b \tag{4.7}$$

where B denotes the set of G-cell boundaries intersected by Ψ^{N_i}, and λ_b represents the number of VSegments crossing the boundary line segment b. A larger value of $\Omega(\Psi^{N_i})$ indicates that Ψ^{N_i} passes through highly congested G-cells. Consequently, this increases the corresponding weight $W(\Psi^{N_i}, T_j)$, thereby assigning higher routing priority to Ψ^{N_i} compared to segments in less congested regions.

If the VSegment Ψ^{N_i} is assigned to T_j, the cost of establishing the VSegment-to-pin connection is denoted by $\chi(\Psi^{N_i}, T_j)$. When the HSegments of Ψ^{N_i} are already routed, a Maze Router is employed to perform the segment-to-segment connection within a single G-cell. Otherwise, the Maze Router is used to establish a path between the pin and Ψ^{N_i}. If no feasible path exists, the vertex (Ψ^{N_i}, T_j) and its adjacent edges are removed from the SCG. If a valid path is found, its cost is defined in Equation 4.8.

$$\chi(\Psi^{N_i}, T_j) = w_1 \cdot \eta_1 + w_2 \cdot \eta_2 + w_3 \cdot \eta_3 \tag{4.8}$$

where,

- η_1: number of detailed routing grid edges crossed in *preferred* routing direction,
- η_2: number of detailed routing grid edges crossed in *non-preferred* routing direction,
- The variable η_3 counts the vias used within the path.

The parameters w_1, w_2, w_3 are the corresponding weights, chosen such that $w_1 < w_2$ to discourage the use of non-preferred routing, and w_3 is assigned the highest value since vias are the most critical resource.

The stitch penalty $\xi(\Psi^{N_i}, T_j)$ associated with assigning the VSegment Ψ^{N_i} to the target pin T_j is defined in Equation 4.9:

$$\xi(\Psi^{N_i}, T_j) = w_4 \cdot \eta_{SP} + w_5 \cdot \eta_{VV} \tag{4.9}$$

where

- η_{SP} denotes the number of short polygons generated in VSegment-to-pin connection
- η_{VV} denotes the number of via violations arising in the VSegment-to-pin connection.

The weights w_4 and w_5 reflect the relative severity of these violations, with $w_4 < w_5$, since a via violation causes greater distortion than a short polygon violation.

To illustrate the method, let us consider a vertical segment Ψ^{N_i} spanning 3 G-cells, each with 4 HTracks. Thus, the segment length is $L(\Psi^{N_i}) = 3 \times 4 = 12$. The average segment length of the panel is $avgSegL = 10$. Since $L(\Psi^{N_i}) \geq avgSegL$ and the track T_j lies within a stitch critical region, the priority function is

$$f(\Psi^{N_i}, T_j) = (1 + r)^{\sigma}, \quad \text{where } r = \frac{12}{10} = 1.2,\ \sigma = 2,$$

giving

$$f(\Psi^{N_i}, T_j) = (1 + 1.2)^2 = 2.2^2 = 4.84.$$

Next, suppose Ψ^{N_i} passes through $|B| = 2$ G-cell boundaries, with $\lambda_{b_1} = 3$ and $\lambda_{b_2} = 5$. Then,

$$\Omega(\Psi^{N_i}) = \frac{1}{2}(3 + 5) = 4.$$

For the VSegment-to-pin connection cost, assuming the Maze Router finds a feasible path with $\eta_1 = 6$ (preferred edges), $\eta_2 = 3$ (non-preferred edges), and $\eta_3 = 2$ (vias). With $w_1 = 1$, $w_2 = 2$, $w_3 = 5$, we compute

$$\chi(\Psi^{N_i}, T_j) = 1 \cdot 6 + 2 \cdot 3 + 5 \cdot 2 = 6 + 6 + 10 = 22.$$

For the stitch penalty, suppose the connection generates $\eta_{SP} = 1$ short polygon and $\eta_{VV} = 2$ via violations. With $w_4 = 2$, $w_5 = 4$, we obtain

$$\xi(\Psi^{N_i}, T_j) = 2 \cdot 1 + 4 \cdot 2 = 2 + 8 = 10.$$

Finally, let $\alpha_1 = 3$. The overall weight from Equation 4.5 is

$$W(\Psi^{N_i}, T_j) = f(\Psi^{N_i}, T_j) \cdot L(\Psi^{N_i}) + \alpha_1 \cdot \Omega(\Psi^{N_i}) - \chi(\Psi^{N_i}, T_j) - \xi(\Psi^{N_i}, T_j),$$

$$W(\Psi^{N_i}, T_j) = 4.84 \cdot 12 + 3 \cdot 4 - 22 - 10 = 58.08 + 12 - 22 - 10 = 38.08.$$

Thus, the weight $W(\Psi^{N_i}, T_j) = 38.08$ indicates a relatively high routing priority for assigning Ψ^{N_i} to track T_j.

Consider the candidate track T_j that lies *outside the stitch critical region*. For the same segment Ψ^{N_i} and parameters as in the previous example ($L(\Psi^{N_i}) = 12$, $avgSegL = 10$, $\alpha_1 = 3$, $\Omega(\Psi^{N_i}) = 4$, $\eta_1 = 6$, $\eta_2 = 3$, $\eta_3 = 2$, $w_1 = 1$, $w_2 = 2$, $w_3 = 5$, $w_4 = 2$, $w_5 = 4$), the priority factor is given by $f(\Psi^{N_i}, T_j) = 1$ (by Equation 4.6), and

$$\chi(\Psi^{N_i}, T_j) = 1 \cdot 6 + 2 \cdot 3 + 5 \cdot 2 = 22.$$

First, let us isolate the effect of the priority factor f. Keeping the stitch penalty identical to the previous case, $\xi(\Psi^{N_i}, T_j) = 10$, Equation 4.5 gives

$$W(\Psi^{N_i}, T_j) = 1 \cdot 12 + \alpha_1 \cdot 4 - 22 - 10 = 12 + 12 - 22 - 10 = -8.$$

This clearly highlights the sole impact of the priority boost f: removing the stitch-region multiplier results in a substantial drop from 38.08 to -8.

Next, let us consider a more realistic outside-SCR penalty. Since stitch-related violations typically decrease outside the stitch critical region, we may take $\eta_{SP} = 0$ and $\eta_{VV} = 0$, which yields $\xi(\Psi^{N_i}, T_j) = 0$. In this case,

$$W(\Psi^{N_i}, T_j) = 1 \cdot 12 + \alpha_1 \cdot 4 - 22 - 0 = 12 + 12 - 22 = 2,$$

This value is much lower than the inside-SCR case (38.08). It shows that $f(\Psi^{N_i}, T_j)$ gives higher priority to longer VSegments inside the stitch critical region, even when stitch penalties become small outside the SCR.

4.6.1.2 Decomposition of Segments and Routing of the Delayed Segments

The *Delayed_Segment_List* contains pairs of VSegments and their associated tracks that must be routed within the *stitch critical region*. For these segments, routing is carried out by decomposing and rearranging them across tracks to minimize short polygon formation.

Consider for example, a segment Ψ^{N_1} from *Delayed_Segment_List* with track choice T_3, along with another segment Ψ^{N_2} from the same list. When assigning Ψ^{N_1} to T_3, multiple scenarios may arise where stitch-line violations could occur. To address these, Figure 4.6 illustrates four distinct solution strategies that resolve all possible following cases.

- **Case I** (Figure 4.6 I.a–I.b): Ψ^{N_1} on T_3 creates a short polygon at $End1$. Introducing a new via resolves this issue.
- **Case II** (Figure 4.6 II.a–II.c): Assigning segments on the chosen VTracks causes a short polygon at $End1$ of Ψ^{N_1}. Segment decomposition followed by rerouting with new vias removes the short polygon.
- **Case III** (Figure 4.6 III.a–III.c): A short polygon occurs at $End1$ of Ψ^{N_1}. Decomposing the segment and rerouting with a via eliminates the violation.
- **Case IV** (Figure 4.6 IV.a–IV.c): Short polygons appear at both $End1$ and $End2$ of Ψ^{N_1}. Decomposition and rerouting with new vias resolve the violations.

4.6.1.3 Assignment of Partial Vertical Segments

While computing the weight W, a particular track option for a Vertical Segment is discarded whenever maze routing fails to identify a feasible path connecting the segment to its pin. As a result, the set of permissible tracks for that VSegment becomes smaller. If every possible track is excluded, the segment cannot be routed on the present layer.

During this stage, we carry out a *partial assignment* for these vertical segments. The MWIS-based strategy is reused, but a revised weight function W' is employed. Here, only the vertical segment is mapped to a VTrack, and its corresponding horizontal segments are postponed to the later phase that handles horizontal segment

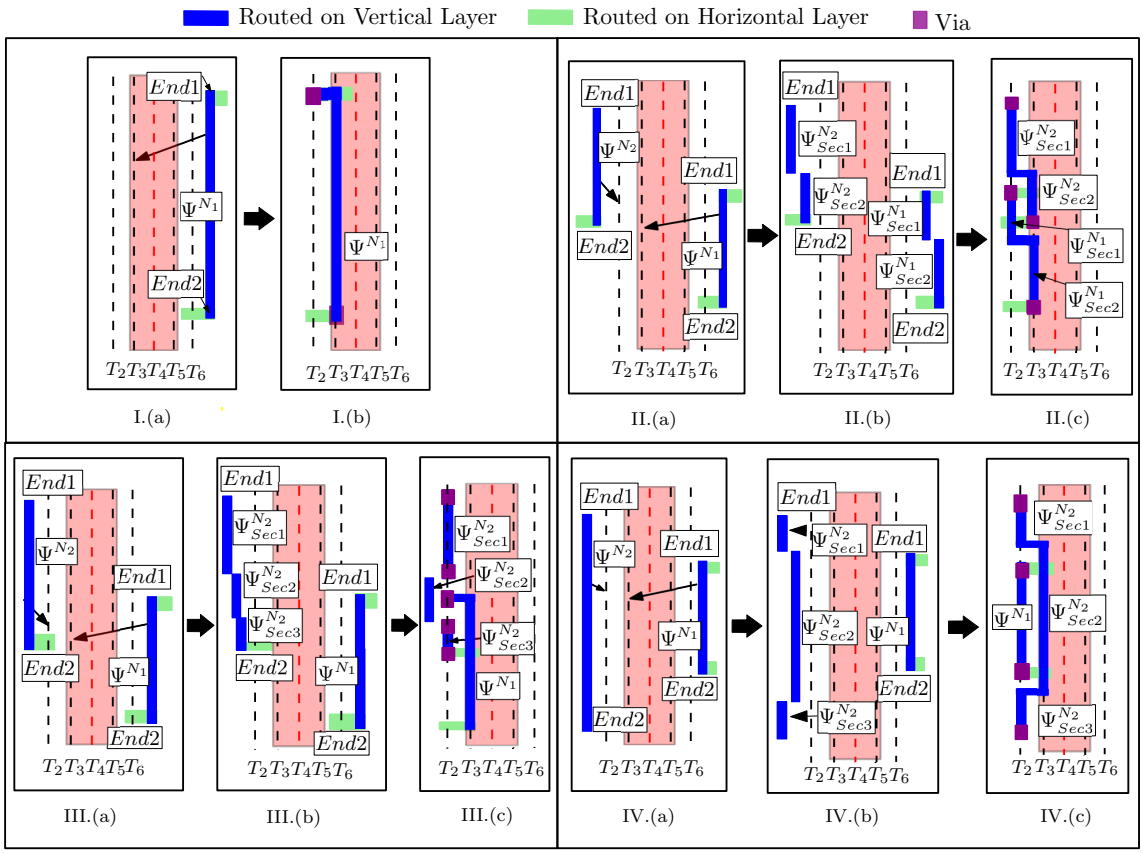


Figure 4.6: Segment splitting and routing of delayed Segments under various scenarios: – Case I: (a) Placing Ψ^{N_1} on T_3 leads to a short–polygon issue at *End*1; (b) introducing a new via resolves the problem. – Case II: (a) The selected VTracks cause a short polygon near *End*1; (b) the segment is divided; (c) the resulting pieces, routed using additional vias, remove the violation. – Case III: (a) The assigned VTracks generate a short–polygon violation at *End*1; (b) the segment is split; (c) routing the split parts with a new via clears the error. – Case IV: (a) The chosen VTracks produce short polygons at both *End*1 and *End*2; (b) the segment is broken into smaller portions; (c) routing these portions using a new via resolves both violations.

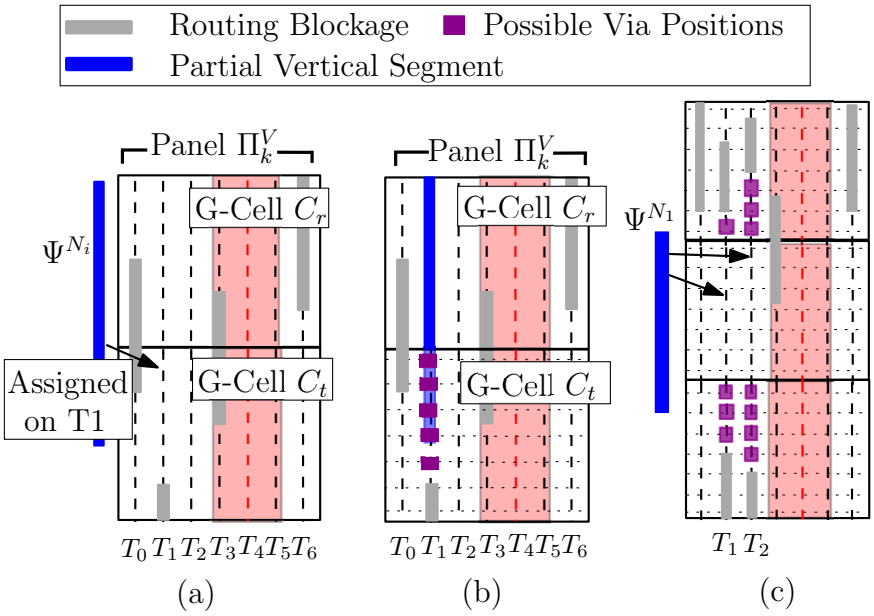


Figure 4.7: Assignment of Partial Vertical Segment: (a) The vertical segment Ψ^{N_i} is partially assigned on T_1;; (b) Candidate via sites for linking to a Horizontal Segment; (c) Feasible via locations at both endpoints of Ψ^{N_1} when mapped to either T_1 or T_2.

routing for MCMPNs. Consequently, W' excludes the vertical segment-to-pin connection cost, since the location of the via has not been fixed at this point.

Figure 4.7(b) illustrates the allowable via locations for a partially routed VSegment Ψ^{N_i} placed on track T_1. Each candidate via shares the same x-coordinate as the selected VTrack, while its y-coordinate is finalized during the HSegment routing stage. Tracks that fall within the *stitch-unfriendly region* are omitted from partial VSegment assignment to prevent the creation of short polygons.

The modified weight W' is defined in Equation 4.10, which incorporates the number of possible via positions for horizontal connections:

$$W'(\Psi^{N_i}, T_j) = L(\Psi^{N_i}) + \alpha_1 \cdot \Omega(\Psi^{N_i}) - \xi(\Psi^{N_i}, T_j) + \alpha_2 \cdot \upsilon(\Psi^{N_i}, T_j) \tag{4.10}$$

where $\upsilon(\Psi^{N_i}, T_j)$ is the minimum number of possible via positions available at the two ends of the segment.

For example, consider a VSegment Ψ^{N_1} that can be assigned either to track T_1 or to track T_2 as shown in Figure 4.7(c). When Ψ^{N_1} is assigned to T_1, maze routing finds only one valid via position at the left end, and three valid via positions at the right end. Hence,

$$\upsilon(\Psi^{N_1}, T_1) = \min(1, 3) = 1$$

This indicates that although multiple options exist at the right end, the bottleneck at the left end restricts the routing flexibility to just one possible connection.

When Ψ^{N_1} is assigned to T_2, maze routing allows three valid via positions at the left end and four valid via positions at the right end. Hence,

$$\upsilon(\Psi^{N_1}, T_2) = \min(3, 4) = 3$$

In this case, routing has significantly more flexibility since both ends of the segment provide multiple valid via positions.

According to the modified weight function W', the assignment of Ψ^{N_1} to T_2 is preferred because it results in a higher value of $\upsilon(\Psi^{N_1}, T_j)$, reflecting a greater number of feasible horizontal connections during the later HSegment routing phase.

4.6.2 Routing of horizontal segments in MCMPNs

The horizontal segments of MCMPNs are handled using Algorithm 4.3. Routing proceeds panel by panel on each HLayer and is divided into two main steps:

- *Step 1:* The horizontal segment assignment is modeled as a problem for finding a Maximum Weighted Independent Set on SCG.
- *Step 2:* Any remaining horizontal segments whose vertical counterparts are already placed are connected to their pins through maze routing.

While constructing SCG and solving MWIS are similar to those for VSegment assignment of MCNs, here we perform both full and partial assignment of HSegments together (6-10 lines of Algorithm 4.3). The weight is given in Equation 4.11.

$$W''(\Psi^{N_i}, T_j) = L(\Psi^{N_i}) + \alpha_1 \cdot \Omega(\Psi^{N_i}) - \chi(\Psi^{N_i}, T_j) - \xi(\Psi^{N_i}, T_j) + \alpha_2 \cdot \upsilon(\Psi^{N_i}, T_j) \quad (4.11)$$

Algorithm 4.3: Routing of HSegments of MCMPNs

```
for each HPanel Π_i in HLayer L_i do
    Find set HSEG of HSegments present in Π_i;
    Construct SCG from HSEG;
    while SCG contains Vertices do
        Find MWIS on SCG to select the best segment track pair (Ψ, T);
        if Corresponding VSegment of Ψ is already Routed then
            Assign HSegment Ψ to T and set Via position;
        else
            Assign HSegment Ψ to T as a partial assignment without
             determining Via position;
        end
        Maze routing for HSegment-pin connection;
        Update SCG ;
    end
    for each unrouted HSegment whose corresponding VSegment is already
     routed do
        Maze routing for VSegment-to-pin Connection;
    end
end
```

No HSegment becomes delayed because the vertex with the maximum benefit is never associated with a track located inside the *Stitch Critical Region*. When (Ψ, T) is selected as the top-benefit choice, segment Ψ is placed on track T. If the vertical portion of Ψ has already been completed, the crossing point between its horizontal and vertical parts determines the via location. Otherwise, when the VSegment remains unrouted, the horizontal part is kept as a partial assignment, and its via site is postponed.

Let us consider a panel Π_i in a horizontal layer L_i containing three HSegments Ψ^{N_1}, Ψ^{N_2}, and Ψ^{N_3}. Suppose the segment conflict graph (SCG) constructed from these segments forms a triangle (clique of size three), since all three HSegments overlap in track usage.

- The MWIS on this SCG selects the HSegment–track pair (Ψ^{N_1}, T_2) as the highest benefit vertex based on Equation 4.11.

- If the corresponding VSegment of Ψ^{N_1} is already routed, we immediately fix the via position at the intersection of Ψ^{N_1} and its VSegment.
- The SCG is updated by removing Ψ^{N_1}. For the remaining HSegments Ψ^{N_2} and Ψ^{N_3}, we again solve MWIS. Suppose Ψ^{N_3} is selected and partially assigned to track T_1 since its VSegment is not yet routed.
- After the SCG becomes empty, the unrouted HSegment Ψ^{N_2} (whose VSegment has since been routed) is connected by maze routing for segment-to-pin connection.

This example illustrates how both *full assignment* (with via position) and *partial assignment* (without via position) are handled in a single unified framework.

Let $|HSEG| = m$ be the number of HSegments in a panel Π_i. Constructing the SCG requires checking pairwise segment conflicts, which takes $O(m^2)$ time. Solving MWIS on SCG is NP-hard in general, but since the SCG is an interval graph (due to geometric track overlap structure), it can be solved in $O(m \log m)$ time. Updating SCG after each assignment requires at most $O(m)$ operations across all iterations. The maze routing for segment-to-pin or VSegment-to-pin connection is polynomial and bounded by $O(|\Pi_i|)$, where $|\Pi_i|$ is the number of grid cells in the panel. Therefore, the overall complexity for routing HSegments in a panel is given in Equation 4.12

$$O(m^2 + m \log m + |\Pi_i|) \tag{4.12}$$

Aggregating over all horizontal panels in the layer, if p is the number of panels, the total complexity is given in Equation 4.13

$$O\left(\sum_{i=1}^{p} \left(m_i^2 + m_i \log m_i + |\Pi_i| \right) \right) \tag{4.13}$$

where m_i is the number of HSegments in panel Π_i.

4.6.3 Routing of MCSPNs having all HSegments in one HPanel

For MCSPNs whose horizontal segments lie completely within a single HPanel, their associated vertical segments remain inside one G-cell and therefore need no indepen-

dent routing stage. These nets are routed using a *multi-layer maze routing* strategy. While processing an HLayer L_i, the algorithm may utilize nearby VLayers and HLayers as auxiliary patch layers to create alternate paths when congestion arises in L_i. The multi-layer maze router maintains good efficiency for the following reasons:

- The total count of MCSPNs in a design is usually small.
- The search space of the maze router is restricted to the HPanel that contains the net.
- Because routing is performed progressively from the lowest layer upward, most MCSPNs are completed in earlier layers, leaving few nets to be lifted to higher layers.
- In the lower patch layers, many tracks become unavailable due to previously completed routes, which naturally limits the maze router's exploration range.

Additional restrictions are imposed on the maze router due to the presence of stitch-lines. To prevent the formation of short polygons, the router is prohibited from changing layers while traversing a *stitch-unfriendly region*. Furthermore, since VTracks along stitch-lines are pre-blocked, the maze router cannot generate vias or cause vertical routing violations. Horizontal patch layers, however, are invoked only when congestion occurs in the current HLayer, because their use requires additional vias. In such cases, the vias or pins of the segment must also be promoted to higher layers, adding further overhead.

Consider an MCSPN whose terminals lie entirely within a single HPanel of width $W = 10$ tracks and height $H = 6$ tracks. The corresponding VSegments for this net are confined within a single G-cell, so no explicit vertical routing is required. The maze router begins on the bottom-most HLayer L_1 and attempts to route the HSegment between the two terminals.

Suppose congestion occurs in L_1 due to previously routed nets occupying 4 of the 10 tracks. The router attempts to find a detour by expanding its search into adjacent patch layers: the vertical layer V_1 and the horizontal layer L_2. If a viable detour is found in V_1, the path is promoted temporarily before returning to L_1 to complete the connection. However, if V_1 is blocked due to pre-routed nets along stitch-lines, the router instead utilizes L_2 as a patch layer, at the cost of inserting vias.

In this example, the maze router is able to connect the two terminals in L_1 by

briefly detouring into L_2, using two vias and an extra horizontal segment. Because the exploration is restricted to a single HPanel, the search space is limited, and the router quickly converges to a feasible solution.

The time complexity of multi-layer maze routing is dominated by the search procedure of the maze router. For a grid of size $W \times H$ tracks and k layers, the worst-case exploration cost is: $T(W, H, k) = O(k \cdot W \cdot H)$. However, in the case of MCSPNs confined to a single HPanel:

- The number of nets m is small (typically $m \ll$ total nets).
- Each net is restricted to its HPanel of bounded size $W \times H$.
- Adjacent patch layers are only invoked in cases of congestion, and their exploration regions are reduced due to pre-blocked tracks.

Thus, the practical complexity reduces to:

$$T_{\text{MCSPN}} = O(m \cdot W \cdot H),$$

where m denotes the count of MCSPNs within a circuit. Since m is usually very small and W, H are bounded by the HPanel dimensions, the multi-layer maze router achieves computational efficiency in practice.

4.6.4 Discussion on the Computational Overhead of the Proposed Framework

The proposed stitch-avoiding detailed routing framework adopts a multi-stage routing strategy to minimize stitch-line violations while maintaining practical routing performance. The routing process is decomposed into several structured stages, including net classification, stitch-avoiding routing of SCNs, routing of the vertical segments of MCNs, routing of horizontal segments, partial segment assignment, and local rip-up-and-reroute.

Although the multi-stage routing strategy introduces additional computational steps compared with conventional detailed routing, several design choices effectively limit the computational overhead. The first design choice is the decomposition of the routing problem according to net span and routing structure. Nets are classified into:

- Single-Cell Confined Nets (SCNs),
- Multi-Cell Single Panel Confined Nets (MCSPNs),
- and Multi-Cell Multi Panel Confined Nets (MCMPNs),

allowing localized routing optimization for different routing scenarios.

Second, routing is performed in a panel-by-panel and layer-by-layer manner, which restricts the search space during track assignment and maze routing. The exploration region for maze routing is therefore significantly smaller than that of unrestricted global detailed routing.

Third, the segment assignment problem is formulated as a Maximum Weighted Independent Set (MWIS) problem on Segment Conflict Graphs (SCGs). Although MWIS is NP-hard in general graphs, the proposed method employs the polynomial-time heuristic used in *RegularRoute* to obtain efficient practical solutions. In addition, several characteristics of practical routing instances further reduce the computational overhead as mentioned in Section 4.6.3. Consequently, although the proposed framework introduces additional routing stages compared with conventional detailed routing approaches, the decomposition of the routing problem, localized optimization, and heuristic-based segment assignment effectively limit the computational overhead. These design choices enable the framework to maintain practical runtime performance while improving manufacturability-aware routing quality. The runtime performance of the proposed framework is evaluated experimentally in Section 4.7.

4.7 Experimental Results

Our proposed stitch-avoiding detailed router is implemented in Java and executed on a Linux system with a 2.40 GHz Intel® Xeon® CPU and 32 GB RAM. For evaluation, we use the Faraday benchmark from the *ICCAD'04 Mixed-Size Placement Benchmarks* [oCADI21, ACR+04], with detailed specifications provided in Table 4.2. The placement of the circuits is performed using NTUplace3 [CJH+08]. The spacing between two stitch-lines is set to 15× the routing pitch, and tracks adjacent to stitch-lines are treated as a *stitch-unfriendly region*.

We construct and evaluate three routing frameworks:

Table 4.2: Faraday benchmark circuits of ICCAD04 [oCADI21]

Circuits	#Layers	Size(μm^2)	#Nets	#Pins
DMA	6	408.4×408.4	13256	73982
DSP1	6	706×706	28447	144872
DSP2	6	642.8×642.8	28431	144703
RISC1	6	1003.6×1003.6	34034	196677
RISC2	6	959.6×959.6	34034	196670

- **Routing Framework I:** Traditional global router *BoxRouter* [CP06] combined with our proposed stitch-avoiding detailed router.
- **Routing Framework II:** Stitch-avoiding global router [SPBSK22] integrated with our proposed stitch-avoiding detailed router.
- **Baseline Routing Framework:** Traditional global router *BoxRouter* [CP06] combined with traditional detailed router *RegularRoute* [ZC12]. As the original implementations of these routers are not publicly accessible for academic research, we implemented them independently. In the baseline detailed router, vertical routing along stitch-lines is blocked to avoid vertical routing violations.

Table 4.3: Routing parameter values

Parameter	Value
α_1	2
α_2	3
σ	2
w_1	1
w_2	$3w_1$
w_3	$2w_2$
w_4	$2w_3$
w_5	$5w_3$

The proposed framework uses several parameters in the stitch-avoiding weight functions, including: $\alpha_1, \alpha_2, \sigma, w_1, w_2, w_3, w_4, w_5$, which control congestion prioritization, stitch penalties, via penalties, segment flexibility, and preferred routing behavior.

The parameter values were empirically selected using repeated benchmark-driven validation to obtain stable routing quality across all benchmark circuits. The routing parameters used in our experiments are summarized in Table 4.3. The relative ordering of these parameters reflects the physical severity of different routing violations:

- via violations are penalized more heavily than short polygons,
- non-preferred routing is penalized more than preferred routing,
- and stitch-critical routing receives additional penalty weighting.

Experimental observations indicate that moderate parameter variations do not significantly degrade routing quality, demonstrating reasonable robustness of the proposed framework. Moreover, the multi-objective weight functions combine congestion awareness, stitch-aware penalties, routing flexibility, and connection feasibility, which together stabilize routing behavior across varying benchmark characteristics. The strong consistency of experimental improvements across all benchmark circuits further indicates that the routing framework is not highly sensitive to minor parameter fluctuations.

Table 4.4 presents a comparison of the three routing frameworks. Here, $\#VV$ denotes the number of via violations, $\#SHP$ the number of short polygons, $\#VRV$ the number of vertical routing violations, and $\%Rout.$ represents the routability percentage. From Table 4.4, we observe that Routing Framework I achieves reductions of 16.18% in via violations and 57.71% in short polygon violations, while Routing Framework II achieves reductions of 50.54% and 43.96% in via violations and short polygon violations, respectively, when compared to the *Baseline Routing Framework*.

Since the placement tools used for the Faraday benchmark suite were not disclosed in [LFC15] and [PBSK17], direct comparison with their results is not possible, as placement significantly impacts stitch-line violations. Furthermore, both Routing Frameworks I and II demonstrate improved routability and lower CPU time compared to the *Baseline Routing Framework*.

Table 4.4: Experimental result showing the effectiveness of our Proposed Stitch-avoiding Detailed Router

	Metrics	Circuits					Ratio
		DMA	DSP1	DSP2	RISC1	RISC2	
Baseline Routing Framework: BoxRouter [CP06] and Regular-Route [ZC12]	$\#VV$	3589	7469	7410	10799	3949	1.00
	$\#SHP$	1626	3496	3462	5246	3183	1.00
	$\#VRV$	0	0	0	0	0	-
	$\%Rout.$	95.94	96.56	96.69	95.92	96.09	1.00
	CPU(s)	421	1419	1296	3016	2944	1.00
Routing Framework I: BoxRouter[CP06] and our proposed Stitch-avoiding Detailed Router	$\#VV$	3546	5568	5462	9536	9586	0.8382
	$\#SHP$	600	1374	1603	2315	2312	0.423
	$\#VRV$	0	0	0	0	0	-
	$\%Rout.$	96.87	96.08	95.81	96.61	97.31	1.0031
	CPU(s)	351	1156	1003	2704	2660	0.8656
Routing Framework II: Stitch-avoiding Global Router [SPBSK22] and our proposed Stitch-avoiding Detailed Router	$\#VV$	2153	3862	3633	4990	5244	0.4946
	$\#SHP$	847	1867	2022	3078	3054	0.5604
	$\#VRV$	0	0	0	0	0	-
	$\%Rout.$	97.26	97.47	97.89	96.79	97.37	1.0115
	CPU(s)	353	1139	1005	2690	2606	0.8567

4.8 Conclusion

This chapter presents a stitch-avoiding detailed router designed to effectively handle stitch-line violations in MEBL. To mitigate stitch-line violations, nets are categorized based on their characteristics, and a multi-step routing strategy is applied to generate solutions with substantially fewer violations. Experimental results demonstrate significant reductions in stitch-line violations for both *Routing Framework I* and *Routing Framework II* compared to the *Baseline Routing Framework*. In particular, *Routing Framework II* shows the best performance, as it integrates a stitch-avoiding global router with our proposed stitch-avoiding detailed router, providing a more comprehensive solution.

While our current framework focuses on routing improvements, certain via violations arising from fixed pins remain unavoidable during the routing phase. Addressing these violations requires coordination with the placement stage, highlighting a natural extension for future work. Additionally, the proposed approach can be further evaluated by integrating alternative routing frameworks or heuristics, which could provide broader insights into stitch-line violation mitigation across different circuit designs. Overall, the presented router establishes a strong foundation for more efficient and reliable MEBL routing, while pointing toward future enhancements in placement-aware routing and cross-layer optimization.

Rectilinear Steiner Minimum Tree (RSMT) construction is a fundamental component of VLSI routing and forms the basis of the stitch-avoiding global and detailed routing frameworks presented in the previous and present chapters, namely Chapters 3 and 4. Since the quality of the routing solution is strongly influenced by the underlying RSMT, efficient and high-quality RSMT construction is crucial for both the proposed MEBL routing framework and VLSI routing in general. As discussed in Section 2.8, the limitations of existing RSMT construction methods motivate the use of machine learning for accurate, scalable, and high-quality RSMT construction. Therefore, this thesis develops two machine learning-based RSMT construction methodologies, presented in the next two chapters (Chapter 5 and 6), for single RSMT construction and concurrent construction of multiple RSMTs, respectively.

CHAPTER

FIVE

CONSTRUCTING RECTILINEAR STEINER MINIMUM TREE WITH CONDITIONAL GENERATIVE ADVERSARIAL NETWORK

Contents

5.1 Introduction **137**

5.2 Motivation **139**

5.3 Preliminaries and Problem Formulation **140**

5.3.1 Preliminaries 140

5.3.2 Problem Formulation 141

5.4 Proposed RSMT Construction Methodology **141**

5.4.1 Overview 141

5.4.2 Image Encoding Mechanisms 143

5.4.3 cGAN Model Architecture for RSMT Construction 153

5.4.4 Loss Functions 155
5.5 Experimental Setup **156**
5.5.1 Generation of Training Dataset 158
5.5.2 Generation of Test Dataset 159
5.5.3 Selection of Baseline Methods 159
5.6 Experimental Results **160**
5.6.1 Computational Cost of the Proposed Framework 164
5.6.2 Generalization Capability of the Proposed Framework 165
5.6.3 Scalability of the Proposed Framework 166
5.6.4 Prediction Reliability 166
5.6.5 Model Interpretability 167
5.7 Concluding Remarks **168**

5.1 Introduction

A *Rectilinear Steiner Minimum Tree* (RSMT) connects a set of d points (the pins of a net) in the 2D plane with the minimum possible total length, potentially introducing extra Steiner points to shorten the routing path. The *degree* is defined as the pin count of a net. An RSMT minimizes the total wirelength measured in the L_1 (Manhattan) distance metric and serves as a backbone for many popular global routing algorithms.

RSMT construction is an important and critical problem in VLSI design with wide-ranging applications. During the initial phases of physical synthesis—including floorplanning, interconnect planning, and placement—RSMTs serve as a basis for estimating wirelength, predicting routing congestion, and evaluating interconnect delay. During routing, they provide the underlying topology for each net. Consequently, efficient RSMT construction is a critical component of modern global routers [CLYP09, SPBSK22]. However, computing an optimal RSMT of a net is NP-hard [GJ77].

In [Han66], Hanan proved that an optimal RSMT always exists on the *Hanan grid*, formed by connecting vertical and horizontal line segments passing through each d pins of a net, as illustrated in Figure 5.1. The Hanan grid contains d^2 grid points. The solver `GeoSteiner` [JWWZ18] can generate optimal RSMTs, but its runtime grows exponentially with the net degree. In contrast, `FLUTE` [CW07] is a widely used heuristic that constructs RSMTs very efficiently and is incorporated into many cutting-edge global routers [CLYP09, SPBSK22]. FLUTE employs a pre-computed lookup table to generate optimal solutions for nets up to degree 9, and applies a net-breaking heuristic for larger nets, which may lead to suboptimal results. The first sub-exponential algorithm for optimal RSMT construction was introduced in [FLK+20], using dynamic programming together with the bounded treewidth property.

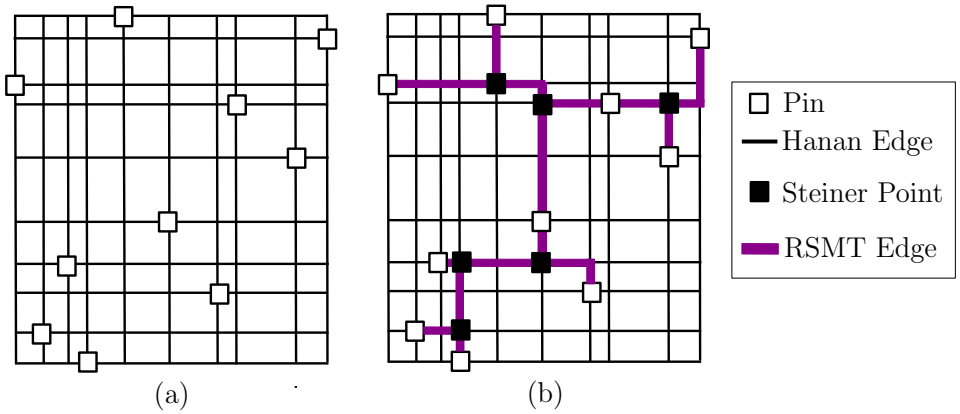


Figure 5.1: (a) Hanan grid with pin positions of a net (b) Optimal RSMT containing only Hanan grid edges and Steiner points

More recently, machine learning–based approaches for RSMT construction have attracted attention. `REST` [LCY21] applies reinforcement learning to construct RSMTs, achieving on average $\leq 0.36\%$ error in wirelength for nets up to degree 50. REST employs a progressive training scheme in which the model for degree-d nets depends on a pre-trained model for degree-$(d-1)$ nets. This dependency causes training time to grow rapidly with net degree; for example, the increase in training time from degree 50 to 51 is about $10\times$ larger than the increase from degree 10 to 11.

The Hanan grid corresponding to a net can be regarded as the input for constructing an RSMT. Since an RSMT containing Hanan grid edges exhibits a distinct

graphical structure, it can naturally be visualized as an image. This motivates viewing RSMT construction as an *image-to-image translation* problem. For example, [JO17] employs image-to-image translation to generate routing solutions for 5-pin nets, while [YZ19] uses a similar approach to forecast routing congestion during placement.

The organization of the remaining chapter is as follows. Section 5.2 provides the motivation to use image-to-image translation for RSMT construction. Section 5.3 defines the preliminaries and problem formulation, and Section 5.4 presents the proposed method. Section 5.5 describes the preparation of the training, validation, and testing datasets. The results of experiments are given in Section 5.6. Section 5.7 concludes the chapter.

5.2 Motivation

Recent advances in image-to-image conversion [IZZE17] with Generative Adversarial Networks (GANs) [GPAM+14] have motivated us to explore GAN-based architectures for RSMT construction. A conventional GAN generates images from random noise, whereas our task requires incorporating the structural information of pins and their Hanan grid to produce a valid RSMT with minimum total wirelength. To achieve this, we adopt a variant known as the Conditional Generative Adversarial Network (cGAN) [MO14, IZZE17], which allows the Hanan grid and pin locations to be encoded as input images and provided as conditions to the generator. Since `GeoSteiner` [JWWZ18] can generate optimal RSMTs—even though with exponential runtime—its solutions can serve as ground truth for effectively training our supervised cGAN model.

The key contributions of this chapter are summarized as follows:

- To our understanding, this work represents the first effective effort to generate an RSMT via image-to-image translation using a cGAN-driven framework.
- We propose two novel image-encoding schemes: one to represent the Hanan grid and pin information of a net as the input image, and the other to represent the corresponding RSMT as the ground-truth image.
- The proposed cGAN-based model achieves an average wirelength error of $\leq$ 0.348% compared with the optimal solution for nets with degree ≤ 50.

5.3 Preliminaries and Problem Formulation

At first, we provide necessary preliminaries, followed by the formally defined problem formulation.

5.3.1 Preliminaries

Table 5.1 gives the list of symbols with their meaning for this chapter.

Table 5.1: List of Symbols and Their Meanings

Symbols	**Meaning**
L_1	Manhattan distance.
inp	Hanan grid input image.
grt	Ground truth RSMT.
gen	Generated RSMT.
d	Degree of a net.
M	Two-dimensional matrix representing Hanan grid.
G	Generator function.
D	Discriminator function.
$distinct_x, distinct_y$	The sets of unique x- and y-coordinates of pins sorted in ascending order.
df_x	The distribution factor of columns of M is the ratio between the number of even-indexed columns available and the number of $distinct_x$ values not yet mapped.
df_y	The distribution factor of rows of M is the ratio between the number of even indexed rows available and the number of $distinct_y$ values not yet mapped.
$distinct_x^t, distinct_y^t$	Geometrically transformed sets of unique x and y-coordinates of pins sorted in ascending order.
$length_z(i, i-1)$	The difference between i^{th} and $(i-1)^{\text{th}}$ index values in $distinct_z$.
l_H	The length of the largest Hanan edge.
f_{upsample}	The learned upsampling function.

Continued on next page

Table 5.1 continued from previous page

Symbols	Meaning
$Loss_G$	The loss function of the whole network.
$cGAN$	Conditional Generative Adversarial Network.
$L_{cGAN}(G, D)$	The adversarial loss of the Discriminator and Generator.
$L_2^W(G)$	A weighted reconstruction loss where W is weight matrix.
λ	A Weight to balance adversarial loss and weighted reconstruction loss.
W, c_1, c_2	Weight matrix and its initial components
B	Mini-Batch Size in training.
E	Number of epochs.
N	The number of training data samples.
η_G, η_D	Learning rates of Generator G and Discriminator D.

5.3.2 Problem Formulation

The problem of generating an RSMT through an image-to-image translation framework can be stated as follows:

Given nets with pin coordinate, design an encoding scheme to represent the pin information as an image, and develop a cGAN-based model that performs image-to-image translation for producing the corresponding RSMT with minimum total wirelength.

5.4 Proposed RSMT Construction Methodology

At first, we give a brief overview of our proposed RSMT construction technique as image-to-image translation using a cGAN-based model in Section 5.4.1.

5.4.1 Overview

Our cGAN model for RSMT construction is based on the `pix2pix` [IZZE17] architecture, as shown in Figure 5.2. It is composed of a PatchGAN [IZZE17]-based discrimi-

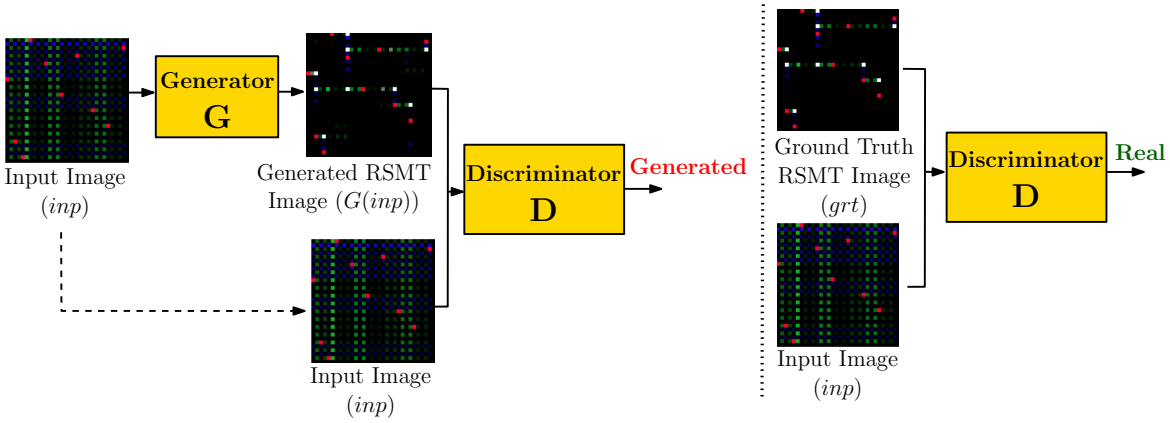


Figure 5.2: Overview of our `pix2pix` based cGAN model with input and ground truth RSMT images

nator and U-Net [RFB15]-based generator. The overall workflow can be summarized as follows:

- The **Generator** G takes as input Hanan grid images inp and generates an RSMT image $gen = G(inp)$. The goal of G is to produce gen that closely matches the ground truth RSMT image grt.
- The **Discriminator** D is trained on pairs of images:
  - (inp, grt) : input with the ground truth RSMT (real pair),
  - (inp, gen) : input with the generated RSMT (fake pair).

  Its objective is to distinguish real pairs from fake ones, while G is trained adversarially so that D is unable to detect that gen is generated.

Formally, the generator learns a mapping

$$G : inp \rightarrow gen, \quad \text{where } gen \approx grt,$$

under the adversarial supervision of D.

In our approach, we focus on constructing RSMTs for nets with degree ≥ 9, since nets with degree ≤ 8 can be generated deterministically. As our cGAN is a supervised learning model, ground-truth data is required for training. To this end, we first generate input images and their corresponding ground-truth RSMT

images by introducing two encoding mechanisms, which are described in Section 5.4.2. Section 5.4.3 and Section 5.4.4 describe the architecture of the proposed model and the associated loss functions.

5.4.2 Image Encoding Mechanisms

To train the generator and discriminator, we first construct input Hanan grid images (*inp*) and ground-truth RSMT images (*grt*). Both *inp* and *grt* are represented as 8-bit RGB images: *inp* encodes the pin locations and Hanan grid information, while *grt* encodes the pins together with the RSMT branches. The key idea is to map image intensity values to represent pins and the lengths of horizontal and vertical Hanan edges. Since pixel intensities in an RGB image must lie within the range $[0, 255]$, we first apply a geometric transformation to normalize the nets, followed by the construction of *inp* and *grt* images based on these transformed nets.

5.4.2.1 Geometric Transformation with Topology Preservation of Nets

We represent a Hanan grid as a two-dimensional square matrix M. In this representation, the pin information of a net is stored at the even indices of M (including index 0), while the odd-numbered indices contain the lengths of the relevant vertical and horizontal Hanan edges. The construction of both *inp* and *grt* images in Figure 5.2 is based on this matrix M, and its dimension is chosen to match that of *inp* and *grt*.

For a degree-d net, if the Hanan grid has dimension $r \times c$ (with $r, c \leq d$), then we define the dimension of M to be at least $2^m \times 2^m$, where 2^{m-1} is the smallest power of two that is no less than $\max(r, c)$. For example, consider the net described in Table 5.2. Since $\max(r, c) = 10$, the dimension of M is taken to be at least 32×32 (For details see Section 5.5).

Table 5.2: Coordinates of the pins of a 10-degree net

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
x	1318	379	1537	4075	2977	634	2617	0	3813	2471
y	0	3597	1627	810	3428	4132	1240	2607	3539	2703

We populate the matrix M with pin information, while the lengths of the horizontal and vertical Hanan grid edges are incorporated through a geometric transformation. The mapping of pin coordinates to M is carried out separately for the x- and y-coordinates, following the same procedure. For illustration, let,

$$distinct_x = \{x_1, x_2, \ldots, x_c\}, \quad x_1 < x_2 < \cdots < x_c,$$

denote the set of distinct x-coordinates of pins sorted in ascending order and

$$distinct_y = \{y_1, y_2, \ldots, y_r\}, \quad y_1 < y_2 < \cdots < y_r,$$

denote the set of distinct y-coordinates of pins sorted in ascending order

To determine their placement in M, we compute a transformed set $distinct_x^t$ and $distinct_y^t$ using Equation 5.1 and Equation 5.3 where df_x and df_y are the distribution factors of columns and rows of M defined in Equation 5.2 and Equation 5.4 respectively. This factor ensures that each value in $distinct_x$ ($distinct_y$) is systematically assigned to a unique even-indexed column (row) of M, preserving geometric consistency across the transformation.

$$distinct_x^t[i] = \begin{cases} 0 & \text{if } i = 0 \\ distinct_x^t[i-1] + 2df_x[i] & \text{otherwise} \end{cases} \tag{5.1}$$

where

$$df_x[i] = \left\lfloor \frac{\text{number of available even-indexed columns in } M \text{ at iteration } i}{\text{number of unmapped elements in } distinct_x \text{ at iteration } i} \right\rfloor. \tag{5.2}$$

We need to calculate $distinct_x^t[i]$ sequentially starting from $i = 1$ to $|distinct_x|$, and after each of i, we have to calculate df_x using Equation 5.2.

Similarly, we need to calculate $distinct_y^t[k]$ sequentially starting from $k = 0$ to

$|distinct_y|$ and after each of k we have to calculate df_y using Equation 5.4.

$$distinct_y^t[k] = \begin{cases} 0 & \text{if } k = 0 \\ distinct_y^t[k-1] + 2df_y[k] & \text{otherwise} \end{cases} \tag{5.3}$$

where,

$$df_y[k] = \left\lfloor \frac{\text{number of available even-indexed rows in } M \text{ at iteration } k}{\text{number of unmapped elements in } distinct_y \text{ at iteration } k} \right\rfloor. \tag{5.4}$$

We encode the pin information into M using Equation 5.5. Let a pin whose x-coordinate is in i^{th} index of $distinct_x$ and y- coordinate is in k^{th} index of $distinct_y$, then the corresponding entry in the matrix M is defined using Equation 5.5.

$$M\left[distinct_y^t[k], distinct_x^t[i]\right] = 255 \tag{5.5}$$

The horizontal (vertical) Hanan edge lengths are geometrically transformed and encoded into the columns (rows) of M using Equation 5.6 (Equation 5.7).

$$M\left[p, q\right] = \left\lfloor (255 - 2^{m-1}) \cdot \frac{length_x(i, i-1)}{l_H \cdot df_x[i] + 1} \right\rfloor \tag{5.6}$$

where,

- p denotes the indices of odd rows in M,
- $q = distinct_x^t[i-1] + 2j - 1 \quad \forall j = 1$ to df_x,
- $length_x(i, i-1) = distinct_x[i] - distinct_x[i-1]$,
- l_H represents the length of the largest Hanan edge.

$$M\left[u, v\right] = \left\lfloor (255 - 2^{m-1}) \cdot \frac{length_y(k, k-1)}{l_H \cdot df_y[k] + 1} \right\rfloor \tag{5.7}$$

where,

- $u = distinct_y^t[k-1] + 2j - 1 \quad \forall j = 1$ to df_y,
- v denotes the indices of odd columns in M,
- $length_y(k, k-1) = distinct_y[k] - distinct_y[k-1]$.

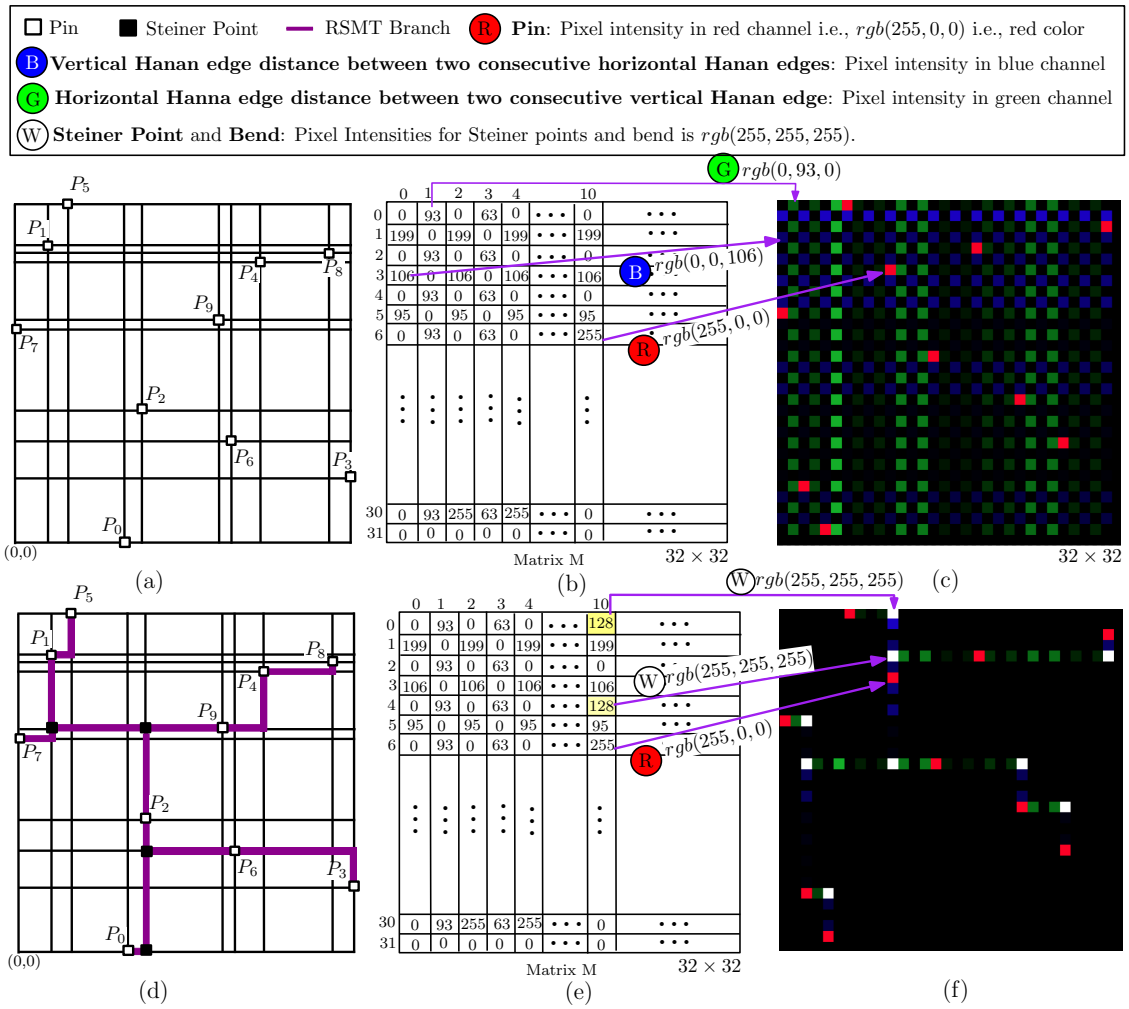


Figure 5.3: Topology-preserving mapping of pin location and length of Hanan edge to a pixel intensity for Input image generation: (a) Hanan grid with pins, (b) corresponding matrix M of size 32×32 (c) corresponding input image encoding (pixels with small intensities may appear to be black however, contain valid information) (d) Ground Truth RSMT (e) Modified Matrix M to represent the Ground Truth RSMT (f) corresponding ground truth RSMT image encoding (pixels with small intensities may appear to be black however, contain valid information).

Figure 5.3(b) illustrates the computation of the matrix M for the example net in Table 5.2. The net contains $d = 10$ pins, and the Hanan grid dimension is $r \times c = 10 \times 10$. To determine the matrix size, we choose the smallest power of two greater than $\max(r, c)$, i.e.,

$$2^{m-1} \geq \max(r, c), \tag{5.8}$$

which gives $m = 5$. Hence, M is initialized as a $2^5 \times 2^5 = 32 \times 32$ matrix with all entries set to zero. The sets of distinct x- and y-coordinates (in ascending order) are:

$$distinct_x = \{0, 379, 634, 1318, 1537, 2471, 2617, 2977, 3539, 4075\},$$

$$distinct_y = \{0, 810, 1240, 1627, 2607, 2703, 3428, 3539, 3597, 4132\}.$$

Using Equation 5.1, we start with $distinct_x^t[0] = 0$ and compute the subsequent transformed coordinates as:

$$distinct_x^t[i] = distinct_x^t[i-1] + 2df_x[i], \tag{5.9}$$

where

$$df_x[i] = \left\lfloor \frac{\text{available even-indexed columns in } M \text{ at iteration } i}{\text{unmapped } distinct_x \text{ elements at iteration } i} \right\rfloor. \tag{5.10}$$

The computations are summarized in Table 5.3. Similarly, for the y-coordinates, following Equation 5.3, we obtain:

$$distinct_y^t = \{0, 2, 4, 6, 10, 14, 18, 22, 26, 30\}.$$

Hence,

$$distinct_x^t = \{0, 2, 4, 6, 10, 14, 18, 22, 26, 30\},$$
$$distinct_y^t = \{0, 2, 4, 6, 10, 14, 18, 22, 26, 30\}.$$

Table 5.3: Computation of transformed x-coordinates

i	$df_x[i]$	$distinct^t_x[i]$
1	$\lfloor 15/9 \rfloor = 1$	2
2	$\lfloor 14/8 \rfloor = 1$	4
3	$\lfloor 13/7 \rfloor = 1$	6
4	$\lfloor 12/6 \rfloor = 2$	10
5	$\lfloor 10/5 \rfloor = 2$	14
6	$\lfloor 8/4 \rfloor = 2$	18
7	$\lfloor 6/3 \rfloor = 2$	22
8	$\lfloor 4/2 \rfloor = 2$	26
9	$\lfloor 2/1 \rfloor = 2$	30

For a pin $P_0(1318, 0)$, the x-coordinate 1318 corresponds to the 3rd index in $distinct_x$, giving $distinct^t_x[3] = 6$, and the y-coordinate 0 corresponds to $distinct^t_y[0] = 0$. Hence, using Equation 5.5:

$$M[distinct^t_y[0], distinct^t_x[3]] = M[0, 6] = 255.$$

Similarly,

$$\begin{aligned} P_1(379, 3597): &\quad M[distinct^t_y[8], distinct^t_x[1]] = M[26, 2] = 255, \\ P_2(1537, 1627): &\quad M[distinct^t_y[3], distinct^t_x[4]] = M[6, 10] = 255. \end{aligned}$$

Other pins can be encoded in a similar manner.

Horizontal Hanan edges between consecutive x-coordinates are encoded using Equation 5.6, where

$$length_x(i, i-1) = distinct_x[i] - distinct_x[i-1]. \tag{5.11}$$

For example,

$$length_x(1, 0) = 379 - 0 = 379.$$

Given the largest Hanan edge length $l_H = 970$, we compute:

$$\begin{aligned} M[p,q] &= \left\lfloor (255 - 2^{m-1}) \cdot \frac{length_x(i, i-1)}{l_H \cdot \lfloor df_x[i] \rfloor + 1} \right\rfloor \\ &= \left\lfloor (255 - 2^{5-1}) \cdot \frac{379}{970 \cdot 1 + 1} \right\rfloor \\ &= \lfloor 239 \times 0.390 \rfloor = 93, \end{aligned}$$

where $p \in \{1, 3, \ldots, 31\}$ and $q = distinct_x^t[0] + 1 = 1$.

The resulting 32×32 matrix M contains:

- 255 at even indices corresponding to pins,
- Transformed Hanan edge lengths at odd rows and columns, and
- Zeros elsewhere.

This matrix is then directly mapped to RGB intensities to generate the input (inp) and ground truth (grt) images.

Algorithm 5.1 describes the entire method of encoding pins and the Hanan grid into matrix M. The required time complexity for constructing M and performing the geometric transformations with topology preservation is $O(d^2)$.

5.4.2.2 Construction of input image (inp)

The one-to-one mapping of M to the input image inp is illustrated in Figure 5.3(b) and Figure 5.3(c), ensuring that the dimensions of inp match those of M. To construct inp, we simply convert the entries of M into corresponding RGB intensity values. Pin locations with the lengths of their horizontal and vertical Hanan edges are encoded in the red, green, and blue channels, respectively. For instance, since pins are located at even-indexed rows and columns with a value of 255, we assign $rgb(255, 0, 0)$ to the corresponding pixels. After applying the proposed geometric transformation, the intensities representing Hanan edge lengths are guaranteed to lie within the range $[0, 255]$.

In Figure 5.4, the entries 255 at even-indexed rows and columns indicate the pin locations, while the values at odd indices represent the lengths of Hanan edges. To construct the input image inp, we convert each entry of M to RGB intensity values:

Algorithm 5.1: Encoding Pins and Hanan Grid into Matrix M

Input: A degree-d net with pin coordinates $(x_j, y_j),\ j \coloneqq 1, \ldots, d$

Output: Two-dimensional matrix M encoding pins and Hanan edges

`// Step 1: Initialize Matrix Dimension`

Let $r \times c$ be the Hanan grid dimension $(r, c \leq d)$;

Set $m \coloneqq \min\{m \in \mathbb{Z}^+ | 2^{m-1} \geq \max(r, c)\}$;

Initialize M as a $2^m \times 2^m$ zero matrix;

`// Step 2: Map Pin Coordinates`

Extract distinct $x-$ and $y-$ coordinates of all the pins and arrange in ascending order: $distinct_x \coloneqq \{x_1, \ldots, x_c\}$ and $distinct_y \coloneqq \{y_1, \ldots, y_r\}$;

Set $distinct_x^t[0] \coloneqq 0$ and $distinct_y^t[0] \coloneqq 0$;

for $i \coloneqq 1$ **to** c **do**

  Compute $df_x[i] \coloneqq \left\lfloor \frac{\#\text{available even-indexed columns in } M \text{ at iteration } i}{\#\text{unmapped elements in } distinct_x \text{ at iteration } i} \right\rfloor$;

  Map transformed coordinates: $distinct_x^t[i] \coloneqq distinct_x^t[i-1] + 2df_x[i]$;

for $k \coloneqq 1$ **to** r **do**

  Compute $df_y[k] \coloneqq \left\lfloor \frac{\#\text{available even-indexed rows in } M \text{ at iteration } k}{\#\text{unmapped elements in } distinct_y \text{ at iteration } k} \right\rfloor$;

  Map transformed coordinates: $distinct_y^t[k] \coloneqq distinct_y^t[k-1] + 2df_y[k]$;

`// Step 3: Encode Pins`

for $j \coloneqq 1$ **to** d **do**

  $(i, k) \coloneqq$ index of (x_j, y_j) in $(distinct_x, distinct_y)$;

  $M\left[distinct_y^t[k],\ distinct_x^t[i]\right] \coloneqq 255$; `// Equation 5.5`

`// Step 4: Encode Horizontal Hanan Edges`

for $i \coloneqq 1$ **to** c **do**

  for $j \coloneqq 1$ **to** $df_x[i]$ **do**

    $p \coloneqq$ odd row indices in M and $q \coloneqq distinct_x^t[i-1] + 2j - 1$;

    $M[p, q] \coloneqq \left\lfloor (255 - 2^{m-1}) \cdot \frac{length_x(i, i-1)}{l_H \cdot \lfloor df_x[i] \rfloor + 1} \right\rfloor$; `// Equation 5.6`

`// Step 5: Encode Vertical Hanan Edges`

for $k \coloneqq 1$ **to** r **do**

  for $j \coloneqq 1$ **to** $df_y[k]$ **do**

    $u \coloneqq distinct_y^t[k-1] + 2j - 1$ and $v \coloneqq$ odd column indices in M;

    $M[u, v] \coloneqq \left\lfloor (255 - 2^{m-1}) \cdot \frac{length_y(k, k-1)}{l_H \cdot \lfloor df_y[k] \rfloor + 1} \right\rfloor$; `// Equation 5.7`

return M;

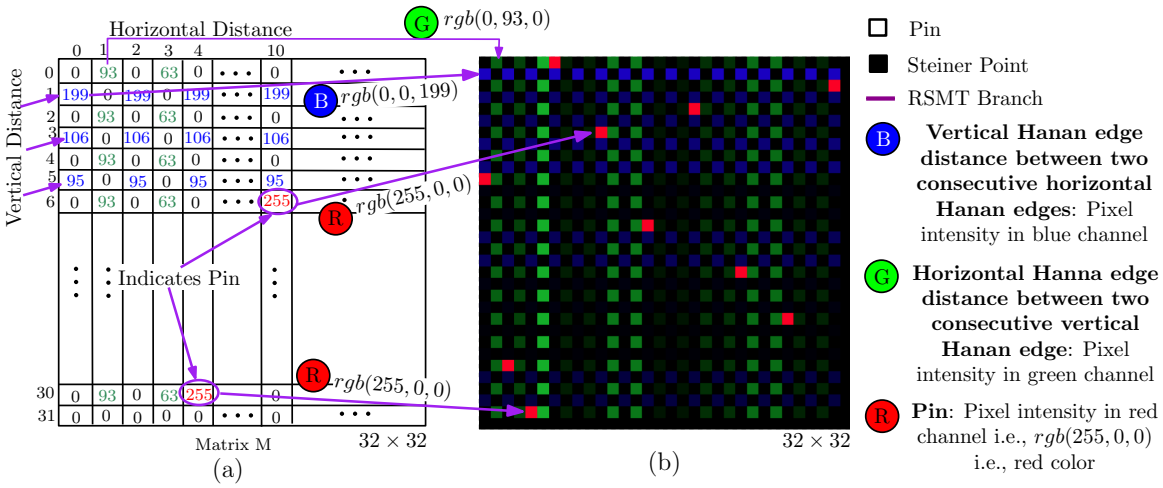


Figure 5.4: (a) Matrix M of size 32×32 (b) corresponding input image encoding (pixels with small intensities may appear to be black, however, contain valid information)

- Pin locations (entries with 255 at even indices) are assigned red intensity: $rgb(255, 0, 0)$.
- Horizontal edges (entries in odd columns) are encoded in the green channel proportional to their length: $rgb(0, g, 0)$, where $g \in [0, 255]$ after geometric transformation.
- Vertical edges (entries in odd rows) are encoded in the blue channel: $rgb(0, 0, b)$, where $b \in [0, 255]$.

After applying the geometric transformation, the resulting RGB image inp preserves the one-to-one mapping from M, with edge lengths scaled to the range $[0, 255]$.

The time complexity of constructing the input image inp from the matrix M is $O(d^2)$, where d denotes the net degree. This is because the matrix M, which has dimensions proportional to the Hanan grid of the net, contains up to $O(d^2)$ entries. Each entry is accessed once to assign an appropriate RGB value based on whether it represents a pin, a horizontal edge, or a vertical edge. Pin placements, located at even-indexed rows and columns, require $O(d)$ operations, while encoding the horizontal and vertical edges between pins may require $O(d^2)$ operations in the worst case. Since every entry in M is processed exactly once, the overall mapping from M to inp scales quadratically with the net degree.

5.4.2.3 Construction of ground truth RSMT image (grt)

To encode an RSMT in grt, we use a ground truth RSMT and the corresponding matrix M of a net as input. All pin information and the lengths of the Hanan grid edges present in the RSMT are stored in M. Steiner points are represented by setting the corresponding entries of M to 128, while all other entries are set to 0. Note that there can be at most $d-1$ Steiner points. Next, the values of M are mapped to RGB intensity values as described previously. Figure 5.3(f) illustrates an example of a ground truth image encoding, where a Steiner point is represented by the intensity $rgb(255, 255, 255)$ (white).

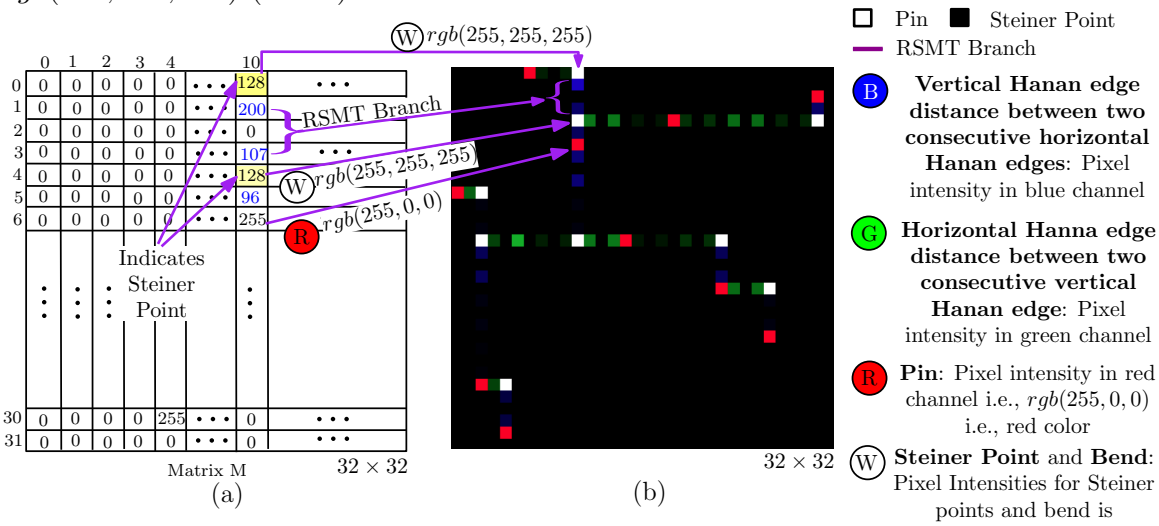


Figure 5.5: (a) Modified Matrix M to represent the Ground Truth RSMT (b) corresponding ground truth RSMT image encoding (pixels with small intensities may appear to be black, however, contain valid information).

In Figure 5.5, entries with 255 correspond to pin locations, the entry 128 represents the Steiner point, and all other entries are set to 0. Next, M is mapped to RGB intensity values to construct the ground truth image grt:

- Pins: $rgb(255, 0, 0)$ (red),
- Steiner points: $rgb(255, 255, 255)$ (white),
- Hanan edges: mapped to green and blue channels according to their lengths as shown in Section 5.4.2.2.

The time complexity of encoding the ground truth image grt is $O(d^2)$. This is because the matrix M has dimensions proportional to the Hanan grid of the net, containing up to $O(d^2)$ entries. Assigning RGB values to each pin or Steiner point requires $O(d)$ operations, while encoding the Hanan edges and counting the number of connections for pins and Steiner points may require $O(d^2)$ operations in the worst case. Since every entry of M is processed exactly once, the overall time complexity for constructing the ground truth image scales quadratically with the net degree.

5.4.3 cGAN Model Architecture for RSMT Construction

The architecture of our U-Net-based generator model is illustrated in Figure 5.6. The generator is an encoder–decoder convolutional neural network (CNN) with skip connections linking encoder and decoder layers of the same spatial size. Let the dimensions of the input, ground truth, and generated images (inp, grt, and gen) be $2^m \times 2^m$. Our U-Net generator consists of m encoder layers and $m+1$ decoder layers. During encoding, the network performs successive downsampling to a bottleneck layer, and during decoding, it up-samples to reconstruct the output while preserving spatial details.

Figure 5.7 shows the architecture of our PatchGAN-based discriminator model, which is a CNN classifier. The discriminator receives either (inp, grt) or (inp, gen), where $gen = G(inp)$ is the image generated by the U-Net. It is trained to distinguish real ground truth images from generated images, evaluating realism at the level of local patches to ensure fine-grained structural accuracy.

Formally, let $inp \in \mathbb{R}^{2^m \times 2^m \times 3}$ denote the input Hanan grid image, $grt \in \mathbb{R}^{2^m \times 2^m \times 3}$ denote the ground truth RSMT image, and $gen = G(inp)$ be the generator output. The generator implements the mapping:

$$G : inp \mapsto gen, \quad gen \approx grt.$$

The encoder extracts hierarchical feature representations, while the decoder reconstructs the spatial structure. Skip connections allow feature maps from encoder layer E_l to be concatenated with decoder layer D_l, preserving spatial information

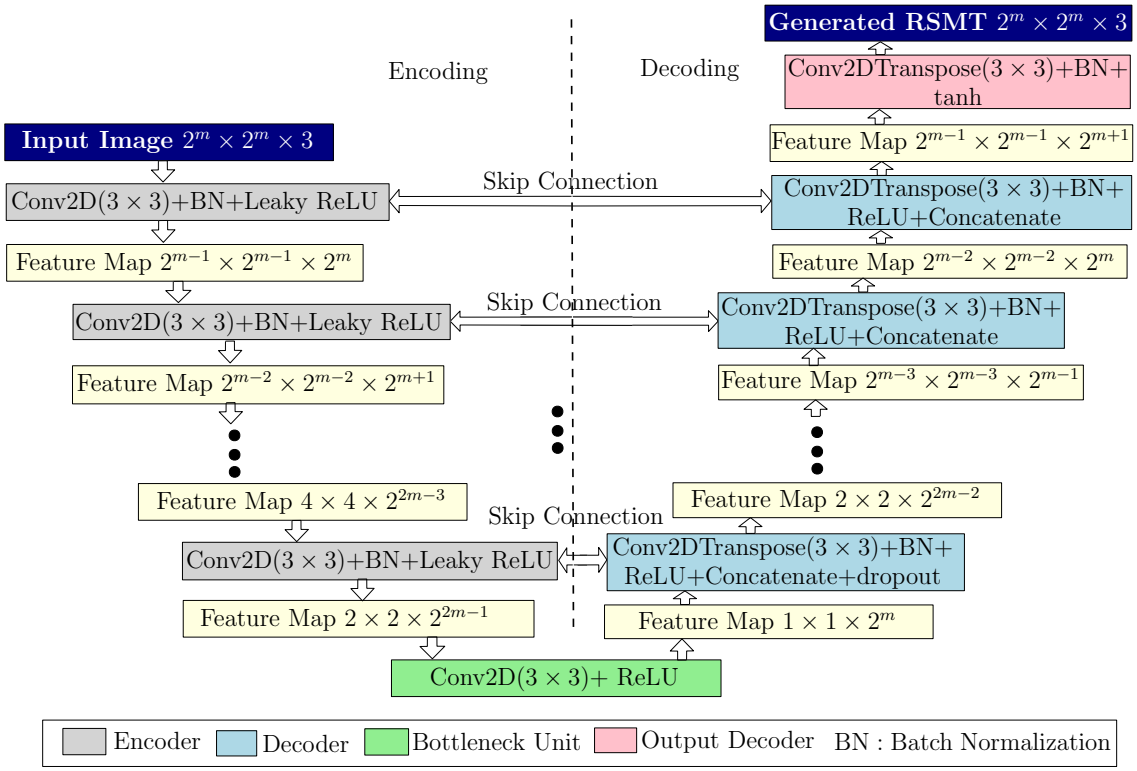


Figure 5.6: Architecture of the U-Net-based generator model

crucial for pin positions and Hanan edges:

$$D_l = f_{\text{upsample}}(D_{l+1}) \oplus E_l,$$

where $\oplus$ denotes concatenation and f_{upsample} is the learned upsampling function.

The PatchGAN discriminator evaluates local image patches rather than the entire image, mapping:

$$D : (inp, img) \mapsto [0,1]^{N \times N}, \quad img \in \{grt, gen\},$$

where each output element indicates the likelihood that the associated patch is real. This design stabilizes adversarial training and emphasizes local RSMT structures. By focusing on small $N \times N$ patches, the discriminator penalizes artifacts at the scale of

pins and edges, ensuring that the generator captures fine-grained RSMT details such as pin connectivity and edge lengths.

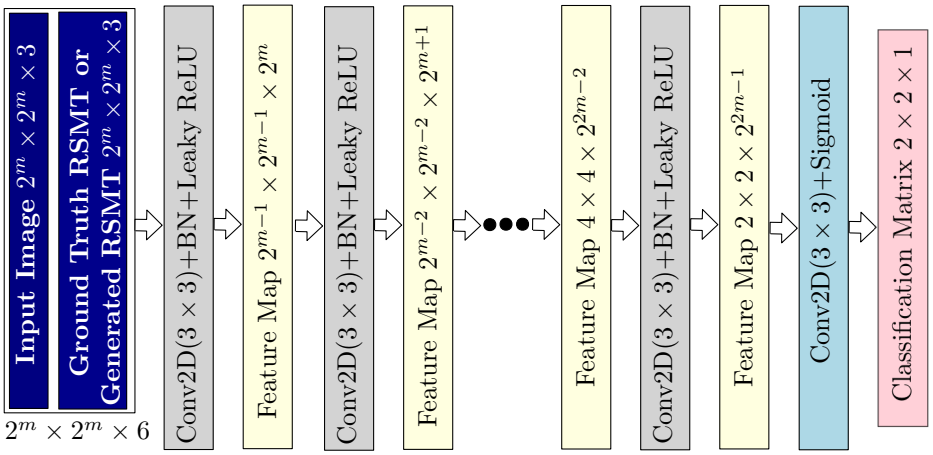


Figure 5.7: Architecture of the PatchGAN-based discriminator model

5.4.4 Loss Functions

To train the generator and discriminator effectively, we use a combination of adversarial and weighted reconstruction losses. The overall objective of our `pix2pix`-based cGAN is

$$Loss_G = \arg\min_G \max_D L_{cGAN}(G, D) + L_2^W(G), \tag{5.12}$$

where $L_{cGAN}(G, D)$ denotes the adversarial loss between the Generator G and the discriminator D.

$$L_{cGAN}(G, D) = \mathbb{E}_{inp,grt}[\log D(inp, grt)] + \mathbb{E}_{inp}[\log(1 - D(inp, G(inp)))]. \tag{5.13}$$

The adversarial loss encourages our generator to produce RSMT images that are indistinguishable from the ground truth, driving global realism in both pin placement and Hanan edge representation. However, because RSMTs are sparse relative to the image background, adversarial loss alone is insufficient to guarantee that the pins and edges are accurately reproduced.

To address this, we introduce a weighted reconstruction loss:

$$L_2^W(G) = \mathbb{E}_{inp,grt}\Big[\big(W \odot (grt - G(inp))\big)^2\Big], \tag{5.14}$$

where W is a weight matrix with higher weights c_1 for even-indexed rows and columns (representing pins and Steiner points) and lower weights c_2 for the rest, with $c_1 > c_2$. This weighted L_2 term ensures that the generator prioritizes the accurate reconstruction of pins and Steiner points while still matching the overall RSMT structure, mitigating the effect of the background pixel imbalance.

The total generator objective can thus be written as

$$\min_G \underbrace{-\mathbb{E}_{inp}[\log D(inp, G(inp))]}_{\text{adversarial loss}} + \lambda \underbrace{L_2^W(G)}_{\text{weighted reconstruction loss}},$$

where λ balances global realism and precise RSMT reconstruction. This combination provides theoretical justification: the adversarial term enforces that generated images are statistically indistinguishable from real RSMTs, while the weighted reconstruction term enforces fidelity to key topological and geometric features.

5.5 Experimental Setup

In our method, the image dimension depends on the degree of the net, i.e., if the image dimension is $2^m \times 2^m$, then our method encodes nets with degrees between $2^{m-2} + 1$ and 2^{m-1}. For example, if the image dimension is 128×128 then our method encodes all the nets having degrees between 33 and 64. The upper bound on the degree is due to our encoding method. We consider the lower bound to improve the training accuracy. If we encode nets with degrees smaller than 33 with an image of dimension 128×128, then the number of pixels holding the information of RSMT in the ground truth image becomes too small, compared with the total number of pixels. Training with this sparse ground truth image decreases the accuracy of the training. As a `pix2pix` model can handle a specific image dimension, if we design and train a particular `pix2pix` based model that can handle a specific image dimension, we can use that model to generate RSMTs of nets with degrees within a particular

Algorithm 5.2: Training Algorithm for Weighted cGAN RSMT Generator

Input: Training dataset $\mathcal{D} = \{(inp_i, grt_i)\}_{i=1}^{N}$, learning rates η_G, η_D, B: batch size , E: #Epochs

Output: Generator G and discriminator D after training

Initialize generator G and discriminator D;
Initialize weight matrix W with $c_1 > c_2$;
for *epoch* $\leftarrow 1$ **to** E **do**
    Shuffle $\mathcal{D}$;
    for *each batch* $\{(inp_i, grt_i)\}_{i=1}^{B}$ **do**
        $gen_i \leftarrow G(inp_i)$; `// Forward pass through generator`
        `// Update Discriminator`
        $L_{cGAN} \leftarrow \frac{1}{B}\sum_i \big[\log D(inp_i, grt_i) + \log(1 - D(inp_i, gen_i))\big]$;
        $\theta_D \leftarrow \theta_D + \eta_D \nabla_{\theta_D} L_{cGAN}$;
        `// Update Generator`
        $L_2^W \leftarrow \frac{1}{B}\sum_i \big(W \odot (grt_i - gen_i)\big)^2$;
        $Loss_G \leftarrow -L_{cGAN} + \lambda\, L_2^W$;
        $\theta_G \leftarrow \theta_G - \eta_G \nabla_{\theta_G} Loss_G$;
    Evaluate on validation set and save checkpoints;
return Trained G and D;

range.

Suppose we have to generate RSMTs for nets with degrees ranging between 9 and 64, then we use three different image dimensions and design and train the `pix2pix` models as follows:

- the first one handles 32×32 images to generate RSMTs for nets with degrees between 9 and 16,
- the second one handles 64×64 images to generate RSMTs for nets with degrees between 17 and 32, and
- the third one handles 128×128 images to generate RSMTs for nets with degrees between 33 and 64.

While the number of convolution layers needed varies with the dimension of the image, the architecture of all the models is the same as shown in Figure 5.6 and Figure 5.7.

5.5.1 Generation of Training Dataset

Our training dataset contains pairs of input (*inp*) and RSMT ground truth (*grt*) images. We consider 8-bit RGB images of dimensions 32×32, 64×64, and 128×128 for nets with degrees within the ranges (9 to 16), (17 to 32), and (33 to 64), respectively. Table 5.4 and Table 5.5 show the total number of nets used for training and validation for each category.

Table 5.4: Training dataset

Category	Dataset	Net Degrees	Image Size	#Nets per degree	Total #Nets
I	TrainDS-I	9 to 16	32×32	2500	20000
II	TrainDS-II	17 to 32	64×64	2500	40000
III	TrainDS-III	33 to 64	128×128	2500	80000

Table 5.5: Validation dataset

Category	Dataset	Net Degrees	Image Size	#Nets per degree	Total #Nets
I	ValDS-I	9 to 16	32×32	2500	20000
II	ValDS-II	17 to 32	64×64	2500	40000
III	ValDS-III	33 to 64	128×128	2500	80000

First, we randomly generate nets for each category as mentioned in Table 5.4 and Table 5.5. We run `GeoSteiner` [JWWZ18] to generate optimal RSMTs for each of these randomly generated nets. Next, we use our encoding mechanisms to generate pairs of input and RSMT ground truth images. Figure 5.8 shows samples of input and ground truth images from our training dataset for category I, where the top row shows the input image and the bottom row shows the corresponding ground truth image.

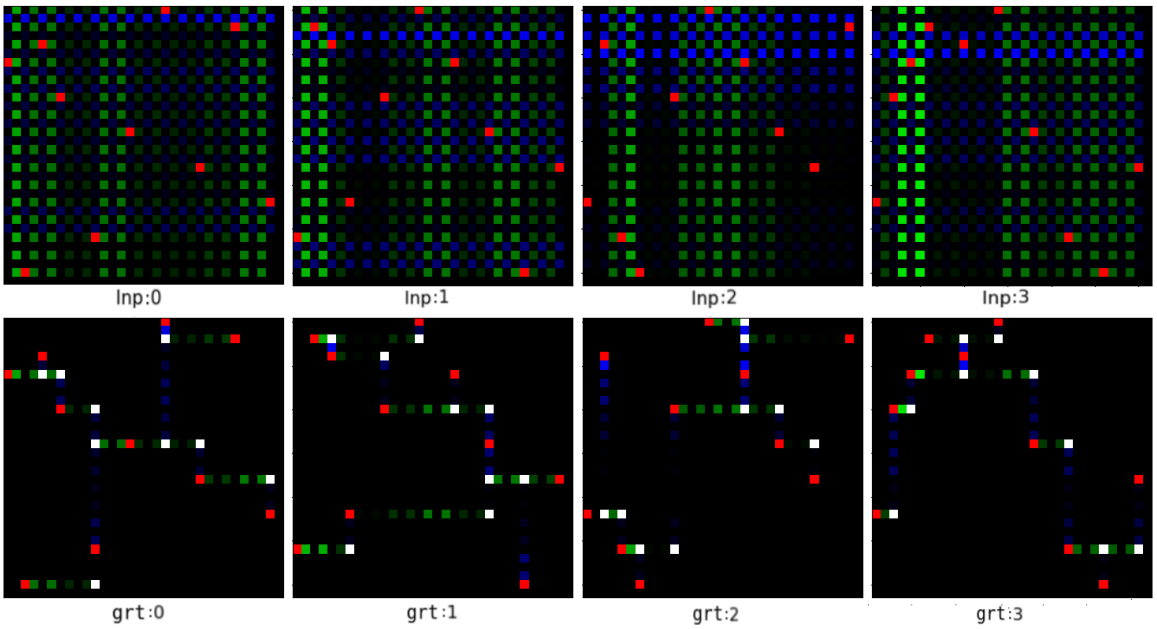


Figure 5.8: Training samples for category I – the upper row shows input images and the corresponding ground truth RSMT images are in the bottom row. (Pixels with small intensities may appear to be black, however, contain valid information.)

5.5.2 Generation of Test Dataset

Table 5.6 presents the details of our testing dataset for each category, and has only the input (*inp*) Hanan grid images. We generate the nets randomly for each degree and use the same encoding method as described in Section 5.4.2 to generate the input images.

5.5.3 Selection of Baseline Methods

The proposed framework was experimentally compared against three well-established RSMT generation methods: GeoSteiner (optimal algorithm) [JWWZ18], FLUTE [CW07], and REST [LCY21] (reinforcement-learning-based approach).

These methods collectively represent exact optimization, fast heuristic routing, and modern machine-learning-based RSMT construction.

Direct comparison with commercial industrial routing tools was not feasible be-

Table 5.6: Test dataset

Category	Dataset	Net Degree	Image Size	#Nets	Category-wise Total #Nets
I	TestDS-I	10	32×32	5000	5000
II	TestDS-II	20	64×64	5000	10000
	TestDS-III	30	64×64	5000	
III	TestDS-IV	40	128×128	5000	10000
	TestDS-V	50	128×128	5000	

cause industrial routing infrastructures are proprietary, commercial RSMT implementations are generally unavailable, and most commercial global routers do not expose standalone RSMT generation modules for independent benchmarking. Nevertheless, the proposed framework demonstrates strong competitiveness relative to established academic and machine-learning-based routing methodologies.

5.6 Experimental Results

We have implemented our `pix2pix`-based cGAN model for RSMT construction using TensorFlow 2.9. We train and test our model on a 64-bit Linux machine with an NVIDIA Tesla P6 GPU with 16 GB of memory. As explained in Section 5.5, we train three models namely $Model_1$, $Model_2$, and $Model_3$ with $TrainDS-I$, $TrainDS-II$, and $TrainDS-III$ training datasets respectively. We train $Model_1$, $Model_2$, and $Model_3$ with batch size 1 for 25 epochs each. We use Adam optimizer with a learning rate of 1×10^{-4} and $\beta = 0.5$ for the generator and learning rate 5×10^{-5} and $\beta = 0.5$ for the discriminator. We consider 50 as the weight in the weighted binary cross-entropy loss of the discriminator. We take $c_1 = 100$ and $c_2 = 10$ to form W.

Figure 5.9 and Figure 5.10 show the changes in the generator and discriminator training losses, respectively. In Figure 5.9, the generator loss decreases significantly with epochs, i.e., the generator is improving at generating RSMTs close to the optimal ground truth. Figure 5.10 depicts the discriminator loss for classifying real RSMT

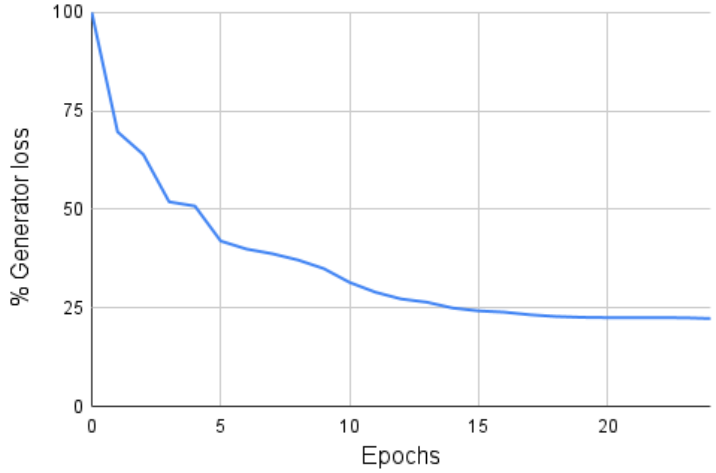


Figure 5.9: Generator loss encountered during the training process

images and generated RSMT images. As we can observe in Figure 5.10, after 14 epochs, the discriminator loss for classifying generated images increases because the generator starts to generate RSMTs close to the optimal.

Figure 5.11 shows the progress of training of our $Model_2$ after epoch 5. Figure 5.11 shows that after 5 epochs, the generated image is not close to the ground truth. However, as the training process continues, the generator gradually learns to produce images that increasingly resemble the ground truth RSMT as the number of epochs grows.

After training, we test our model using the test dataset described in Table 5.6. Figure 5.12 shows generated RSMTs for a few nets of our test datasets. We decode the tree edges and Steiner point information from the generated 8-bit RGB RSMT image.In order to obtain information on RSMT branches from the generated RSMT images, we use majority voting. We consider a branch an RSMT branch if non-black pixels at odd indices of that branch are in the majority. During test dataset construction, we stored the mapping between pin coordinates and image indices. We use these stored mappings to get the coordinates of pins, edges, and Steiner points of the generated RSMTs.

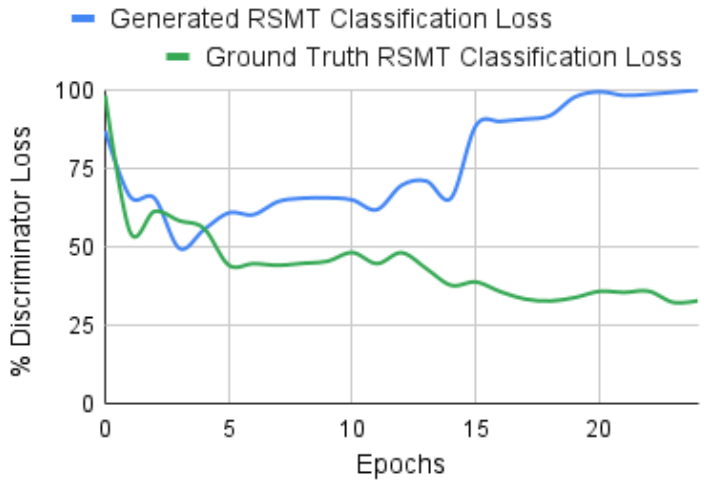


Figure 5.10: Discriminator loss encountered during the training process, where the green line denotes the discriminator loss for classifying real RSMT images and the blue line denotes the discriminator loss for classifying generated RSMT images

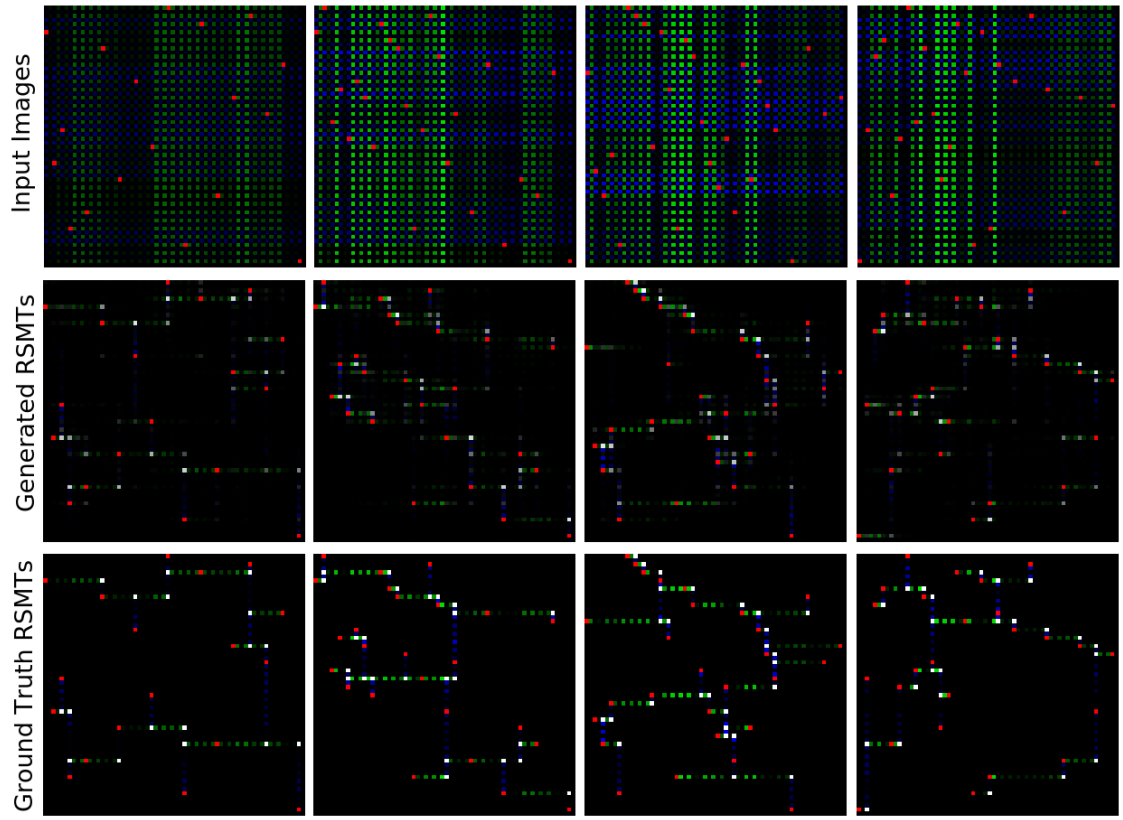


Figure 5.11: Progress of the generator model after 5 training epochs. The upper row contains the input image, the middle row shows the RSMT obtained after 5 epochs, and the lower row presents the actual RSMTs. Pixels with minimal intensity can look black while still encoding valid information.

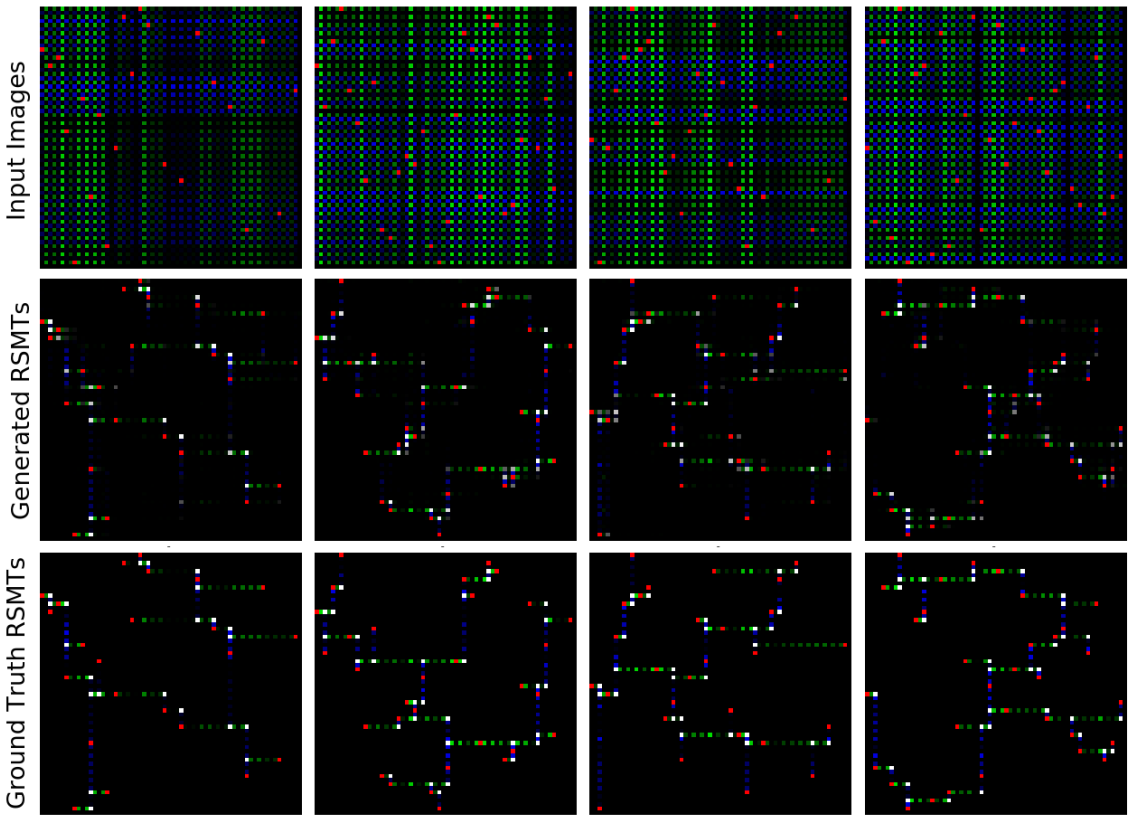


Figure 5.12: Results of Testing. The top row shows input images, the middle row shows generated RSMTs, and the bottom row shows the ground truth RSMTs. Pixels with small intensities may appear to be black. However, they contain valid information.

We compared the wirelength and time to generate the RSMTs obtained by our method with three state-of-the-art methods `GeoSteiner` (optimal) [JWWZ18], `FLUTE` [CW07], and `REST` [LCY21] using the test datasets listed in Table 5.6. For `FLUTE`, we set the accuracy to 18, which is its most accurate setting. For `REST`, we use the 8-transformation version that includes rotating the point set by 0, 90, 180, and 270 degrees, with or without their x and y coordinates being swapped, and take the best solution. Table 5.7 gives the comparison of the RSMT wirelength and Table 5.8 the RSMT generation time. Further, Table 5.8 shows that the time for RSMT generation by our method depends on the input image size. For example, although the

net degree of TestDS-IV is smaller than that for TestDS-V, we use the same image size of 128×128 and thus the RSMT generation times are close.

Table 5.7: Comparison of average percentage error in wirelength of our method with `GeoSteiner` [JWWZ18], `FLUTE` [CW07], and `REST` [LCY21]

Test Dataset	`GeoSteiner` [JWWZ18]	`FLUTE` [CW07]	`REST` [LCY21]	Ours
TestDS-I	0.0	0.0	0.00016	0.00005
TestDS-II	0.0	0.00049	0.00652	0.076
TestDS-III	0.0	0.0556	0.04691	0.04201
TestDS-IV	0.0	0.1898	0.19362	0.1729
TestDS-V	0.0	0.43612	0.38032	0.34801

Table 5.8: Comparison of RSMT Generation Time (sec) of our method with `GeoSteiner` [JWWZ18], `FLUTE` [CW07], `REST` [LCY21]

Test Dataset	`GeoSteiner` [JWWZ18]	`FLUTE` [CW07]	`REST` [LCY21]	Ours
TestDS-I	1.32	0.01	2.53	1.02
TestDS-II	5.12	2.07	3.15	5.17
TestDS-III	12.41	7.96	6.16	6.3
TestDS-IV	24.54	14.19	8.87	11.83
TestDS-V	32.73	19.21	11.45	11.76

5.6.1 Computational Cost of the Proposed Framework

The proposed cGAN-based framework involves computationally intensive training because the generator and discriminator are trained adversarially, large-scale image datasets are required, and high-dimensional convolutional operations are performed repeatedly during training.

In the present work, three separate pix2pix-based cGAN models were trained for different net-degree ranges using 20,000 training samples for degree range 9–16, 40,000

training samples for degree range 17–32, and 80,000 training samples for degree range 33–64.

The models were trained on a 64-bit Linux machine with NVIDIA Tesla P6 GPU having 16 GB of memory using TensorFlow 2.9. However, the computational overhead occurs primarily during the offline training stage. Once training is completed, RSMT generation during inference becomes significantly faster and suitable for practical deployment.

The experimental comparison shows that for higher-degree nets, the proposed framework achieves competitive runtime performance compared to state-of-the-art methods. For example, for degree-50 nets, GeoSteiner requires 32.73 seconds, FLUTE requires 19.21 seconds, REST requires 11.45 seconds, while the proposed framework requires only 11.76 seconds with near-optimal wirelength quality. Thus, the proposed framework trades a higher one-time offline training cost for efficient inference-time RSMT generation and near-optimal routing quality.

5.6.2 Generalization Capability of the Proposed Framework

The proposed cGAN framework was trained using large-scale randomly generated net distributions covering multiple routing topologies and pin configurations. The training datasets include diverse net structures with varying pin locations, different Hanan grid dimensions, varying Manhattan distance distributions, and multiple routing topologies generated through GeoSteiner.

The testing datasets were generated independently from the training datasets and contain unseen net configurations across multiple net-degree categories. Experimental results demonstrate strong generalization capability across unseen routing instances. The proposed framework achieves 0.00005% average wirelength error for degree-10 nets, 0.04201% error for degree-30 nets, and only 0.34801% error for degree-50 nets compared to optimal GeoSteiner solutions.

The topology-preserving image encoding also improves generalization because the model learns structural routing patterns, geometric relationships, and connectivity features rather than memorizing specific coordinate values. Moreover, adversarial training improves the model's ability to generate structurally valid RSMTs for unseen

inputs. Nevertheless, extremely different routing distributions or industrial placement patterns may require additional domain-specific training data, transfer learning, or domain adaptation strategies for optimal performance.

5.6.3 Scalability of the Proposed Framework

The present implementation evaluates the framework for nets with degrees up to 50 because higher-degree nets require larger Hanan grids, larger Hanan grids require higher-resolution image representations, and the encoder-decoder architecture complexity increases with image dimension.

The current framework uses 32×32, 64×64, 128×128 image dimensions for handling progressively larger net-degree ranges. The practical limitation in the current implementation arises primarily from available GPU memory, training time, and computational resource constraints, rather than from the proposed methodology itself. As discussed in the thesis, extending the image dimension to 256×256 would enable handling nets with degrees up to approximately 128 while preserving the same encoding methodology.

Furthermore, the proposed image encoding mechanism and cGAN architecture are inherently scalable because the topology-preserving encoding remains valid for arbitrary Hanan grid sizes, the U-Net generator can be extended by adding encoder-decoder stages, and patchGAN discriminators naturally support larger image resolutions.

The present work therefore establishes the feasibility of image-to-image translation-based RSMT generation, while larger-scale implementations may benefit from higher-memory GPUs, multi-scale GAN architectures, hierarchical image decomposition, or distributed training strategies.

5.6.4 Prediction Reliability

The current framework primarily focuses on accurate RSMT generation, wirelength minimization, and topological correctness rather than probabilistic confidence estimation. The pix2pix-based cGAN architecture generates deterministic routing outputs

conditioned on the input Hanan-grid image. Consequently, explicit uncertainty estimation mechanisms were not incorporated in the present framework. However, several factors indirectly contribute toward prediction reliability:

- adversarial training improves structural consistency,
- weighted loss functions reduce invalid sparse predictions,
- and topology-preserving encoding constrains the output search space.

The low wirelength error observed consistently across all testing datasets further indicates stable prediction behavior. Explicit uncertainty-aware extensions may nevertheless improve reliability for industrial deployment. Such extensions may include Bayesian GAN frameworks [SW17], Monte Carlo dropout[GG16], ensemble-based prediction confidence [LPB17], or probabilistic routing uncertainty estimation [KG17]. These approaches could help identify routing instances where the generated RSMT may deviate significantly from optimal solutions.

5.6.5 Model Interpretability

Like many deep generative models, our proposed cGAN framework learns complex latent routing representations that are not always directly interpretable in explicit algorithmic form. The generator learns routing structures through hierarchical convolutional feature extraction, encoder-decoder spatial transformation, and adversarial optimization against the discriminator.

Although the learned latent features are not explicitly visualized in the present work, several architectural choices improve structural interpretability, topology - preserving image encoding constrains routing geometry, skip connections preserve local spatial information, and patchGAN discriminators emphasize local routing consistency.

The generated RSMTs additionally preserve rectilinear routing structures, Hanan-grid embeddings, and valid pin connectivity, which indicates that the learned features capture meaningful routing characteristics. Furthermore, the gradual improvement observed during training demonstrates that the model progressively learns valid routing topologies from geometric image structures. More detailed interpretability analysis may nevertheless be explored through feature-map visualization, latent-space anal-

ysis, saliency mapping, and explainable-AI techniques for generative routing models. These directions represent valuable future extensions toward understanding deep-learning-driven routing behavior more comprehensively.

5.7 Concluding Remarks

This chapter presents a supervised machine-learning method for constructing an RSMT using image-to-image translation with a cGAN-based model. Two novel image encoding mechanisms have been proposed to form the training and testing dataset. A cGAN-based model is designed, consisting of a U-Net-based generator and a patchGAN-based discriminator. A custom loss function is defined to train the generator properly. After training, the experimental results show that our model generates RSMTs with $\leq 0.348\%$ increase in wirelength on average compared with the optimal solution.

Due to the limitations of the available computational resources, we have used image dimensions of at most 128×128 here. With higher computational resources, this method can easily be extended by considering images of larger dimensions and extending our generator and discriminator model accordingly. For example, if the image dimension is considered to be 256×256, then we can generate RSMTs for nets with degrees up to 128. The method in [WLZ+18] proposed a cGAN model, namely `pix2pixHD`, which can handle images of dimensions 2048×1024. Hence, using larger image dimensions and extending our models to generate RSMTs for nets with higher Hanan grid dimensions is the future scope of this work.

Modern VLSI designs require the construction of RSMTs for a large number of nets. Existing approaches generate each RSMT independently, overlooking the interactions among neighboring nets and limiting opportunities to optimize local edge overlap. Motivated by these limitations, the next chapter (Chapter 6) develops a novel machine learning-based methodology for the concurrent construction of RSMTs of multiple nets, enabling both improved local edge-overlap optimization and efficient RSMT generation.

CHAPTER

SIX

CONCURRENT GENERATION OF RSMTS FOR MULTIPLE NETS USING GRAPH NEURAL NETWORK

Contents

6.1 Introduction **171**

6.2 Motivation and Major Contributions **172**

6.3 Preliminaries and Problem Formulation **174**

6.3.1 Preliminaries 175

6.3.2 Problem Formulation 179

6.4 Overview of Proposed Methodology **180**

6.5 Generation of Dataset **182**

6.5.1 Net Grouping with Fixed Group Size 182

6.5.2 Combined Hanan Grid Graph Construction 185

6.5.3 Edge-Overlap-Minimized Combined RSMT Construction 187

6.6 Our Hybrid GNN Model Architecture and Loss **191**
6.6.1 Input Encoding using Linear Encoders 192
6.6.2 Message Passing Layers 195
6.6.3 Jumping Knowledge Aggregation 198
6.6.4 Global Attention, Virtual Slots, and FiLM Modulation 199
6.6.5 Adaptive Edge-Level Attention 200
6.6.6 Modeling Inter-Net Dependencies 201
6.6.7 Node Classifiers 202
6.6.8 Edge Classification Head 203
6.6.9 Discussion on the Selection of the GNN Architecture 205
6.6.10 Loss Generation 206
6.7 Formation of Individual Predicted RSMTs **207**
6.7.1 Branch-Aware Edge Probability Smoothing 208
6.7.2 Priority-Guided Tree Construction 208
6.8 Experimental Setup **208**
6.8.1 Selection of Hyperparameters 209
6.8.2 Details of Training Dataset 209
6.8.3 Details of Test Dataset 211
6.9 Experimental Results **211**
6.9.1 Generalization Capability of the Proposed Framework 216
6.9.2 Interpretability of Learned Representations 217
6.10 Concluding Remarks **218**

6.1 Introduction

Due to technological advancement, modern ICs are becoming more complex and dense, containing a significantly large number of nets. Due to this high layout density, minimizing the wirelength of nets is one of the primary objectives of the routing phase. As the Rectilinear Steiner Minimum Tree (RSMT) is the shortest route to interconnect the pins of a net in L_1 (Manhattan) distance, many state-of-the-art global routers like [CP06, SPBSK22] use RSMTs to generate the initial routing topologies.

As discussed in Section 2.8 and Section 5.1, according to [Han66], an optimal RSMT only contains a particular subset of edges of the Hanan grid. Figure 6.1 shows a Hanan grid of a net and its optimal RSMT formed only by the edges of the Hanan grid. Finding an optimal RSMT is challenging because the problem is NP-hard [GJ77]. Hence, there are existing techniques that either generate optimal RSMTs in exponential/sub-exponential time or near-optimal RSMTs in polynomial time using either heuristics or machine learning methods.

In Section 2.8, we have classified all the major top-tier methods for RSMT construction into four categories. All existing methods within each category construct RSMTs sequentially, processing one net at a time. As modern ICs contain a large number of nets, concurrent construction of RSMTs for multiple nets has the potential to reduce the construction time of all nets present in an IC, and during concurrent construction, we can also reduce the edge overlap between different nets.

This chapter presents a method for the concurrent construction of RSMTs of multiple nets that reduces the edge overlap and construction time, and also minimizes the total wirelength. This chapter is organized as follows: Section 6.2 presents the motivation and major contributions. Section 6.3 presents the preliminaries and problem formulation. Section 6.4 provides an overview of the proposed method, while Section 6.5 details the dataset generation process. Section 6.6 describes the hybrid GNN architecture and loss function. Section 6.7 shows the formation of individual predicted RSMTs. Section 6.8 outlines the experimental setup. Section 6.9 shows the experimental results, and Section 6.10 gives the concluding remarks.

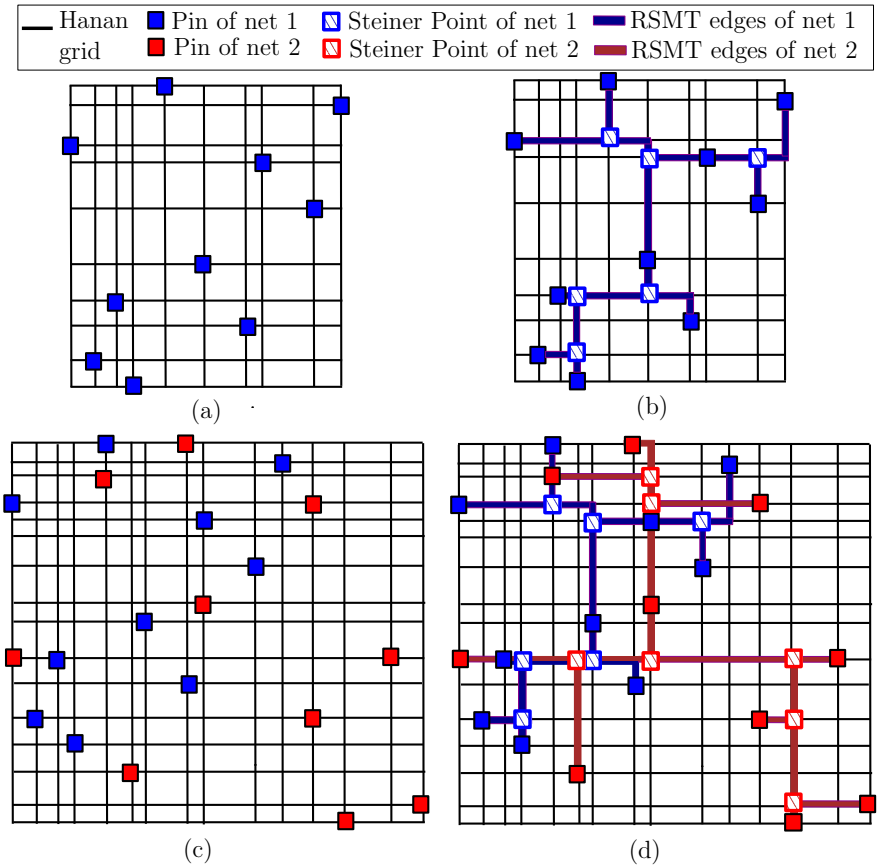


Figure 6.1: (a) Hanan grid with pin positions of a net (b) Optimal RSMT containing only Hanan grid edges and Steiner points (c) Combined Hanan grid with pins of two nets (d) Corresponding combined ground truth optimal RSMTs of two nets

6.2 Motivation and Major Contributions

Concurrent construction of RSMTs of multiple nets further raises the problem complexity. As construction of an optimal RSMT of a single net is still at best sub-exponential [FLK+20], we are motivated to use a machine learning technique for constructing RSMTs of multiple nets concurrently that runs in polynomial time and minimizes the total wirelength error overhead.

In [SBSK25], the Hanan grid graph is used as input to construct the RSMT of a single net. For multiple nets, we need to generate individual Hanan grids and take

their union to form a combined Hanan grid graph containing all pins. To differentiate pins across nets, nodes must have appropriate features. Similarly, edges must have information to indicate their corresponding nets. Some edges may belong to multiple RSMTs, so features must reflect multi-net associations. This combined grid enables concurrent RSMT construction. Thus, both node and edge features are required. To process these features, a Graph Neural Network (GNN) can be employed to construct RSMTs concurrently across multiple nets.

GNNs are specialized neural networks that operate on graph-structured data. They capture dependencies among nodes through their connections, making them suitable for tasks such as node classification, link prediction, and graph classification. Trained on labeled graphs, GNNs learn mappings from node and edge features to corresponding labels by aggregating information from each node's neighborhood. Through iterative message passing and feature updates, they learn both local and global structural representations.

As GNNs are a supervised learning method, we need to generate labeled graph data. The combined Hanan grid graph can be used as input. To form the ground truth, we can use `GeoSteiner` [JWWZ18] to generate individual optimal RSMTs of multiple nets and apply some overlap mitigation technique, and then we can find the union of these edge-overlap-minimized optimal RSMTs. We can label the edges of the combined Hanan grid accordingly. Hence, in this chapter, we propose a method for concurrent construction of RSMTs of multiple nets using a graph neural network that minimizes the overall generation time, group-wise edge overlaps of multiple RSMTs, and wirelength of RSMTs. The key contributions of this work include the following:

- Based on the available literature, this study constitutes the first demonstrated methodology to construct group-wise edge-overlap-minimized RSMTs of multiple nets concurrently using a GNN model.
- To construct the training and validation datasets for concurrent multi-net handling, we proposed two methods:
  1. to form the combined Hanan grid graph as input;
  2. generate edge-overlap-minimized combined RSMT graphs as ground truth. To minimize edge overlap in the ground truth, we proposed an ILP-based method.

- We proposed two novel encoding methods for nodes and edges containing both handcrafted and learnable features.
- We design a hybrid GNN model, and after training, our model generates group-wise edge-overlap-minimized RSMTs of $K = 4$ nets concurrently with approximately 20.32× faster construction time, 0.2599% wirelength error on average compared to the optimal baseline, and also mitigates 98.415% group-wise edge-overlap on average.

The proposed GNN framework in this chapter and the cGAN-based framework presented in Chapter 5 address different routing objectives. The cGAN approach operates on a single net at a time and formulates RSMT construction as an image-to-image translation problem. It generates near-optimal RSMTs by learning geometric routing patterns from Hanan-grid image representations.

In contrast, the proposed GNN framework simultaneously processes multiple nets through a combined Hanan-grid graph representation. This enables explicit modeling of interactions among nets during RSMT construction. Consequently, the GNN framework offers several practical advantages for concurrent routing: simultaneous construction of multiple RSMTs, reduction of group-wise edge overlap during construction, direct modeling of inter-net dependencies, and improved overall construction throughput.

Experimental results demonstrate that the proposed framework achieves approximately 20.32× faster construction compared with sequential baseline approaches while maintaining an average wirelength error of only 0.2599% relative to optimal solutions.

Therefore, the cGAN framework is particularly suitable for high-quality single-net RSMT generation, whereas the GNN framework is designed for concurrent multi-net optimization where interactions among nets must be considered explicitly.

6.3 Preliminaries and Problem Formulation

In this section, first, we describe a few preliminaries, followed by the formulation of the problem statement.

6.3.1 Preliminaries

GNNs operate on graph-structured data, where each node $v \in \mathcal{V}$ is associated with a feature vector $\mathbf{h}_v$, and edges $(i, j) \in E$ are associated with features $\mathbf{e}_{ij}$. Graph Convolutional Network (GCN) is one foundational model that generalizes the concept of convolution to graphs. GCN updates node representations through local neighborhood aggregation. The standard GCN layer, introduced by Kipf and Welling in [Kip16], is defined in Equation 6.1.

$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right) \tag{6.1}$$

where, $\hat{A} = A + I$ is the adjacency matrix with added self-loops, $\hat{D}$ is its degree matrix, $\mathbf{H}^{(l)}$ are the node features at layer l, $\mathbf{W}^{(l)}$ is a learnable weight matrix, and σ is a non-linear activation function.

However, GCNs often struggle to effectively capture complex edge interactions and suffer from vanishing gradients when stacking multiple layers. To address these challenges, the residual gated GCN [BL17] introduces two key ideas:

- Gating mechanisms on edges to modulate message passing dynamically.
- Residual connections combined with Layer Normalization to stabilize deep architectures.

Residual gated GCNs are effective for fine-grained relational tasks, such as molecular property prediction [YSJ+19] and structured prediction [LTBZ15]. Each block performs edge updates from node features, computes gates using node and edge features, aggregates messages through gating, and applies a residual connection with normalization and activation.

In Residual Gated GCNs, for each edge (i, j), a gate $\mathbf{g}_{ij} \in [0, 1]$ is computed as:

$$\mathbf{g}_{ij} = \sigma(f(\mathbf{h}_i, \mathbf{h}_j, \mathbf{e}_{ij})) \tag{6.2}$$

using a function (e.g., an MLP) based on the node features of i and j, as well as the edge feature $\mathbf{e}_{ij}$. Gated message from neighbor j to node i is then computed as $\mathbf{g}_{ij} \cdot \phi(\mathbf{h}_j, \mathbf{e}_{ij})$, where ϕ is a message function. The updated message for node i is

typically an aggregation over all gated neighbor contributions (Equation 6.3).

$$\mathbf{m}_i = \sum_{j \in \texttt{NEB}(i)} \mathbf{g}_{ij} \cdot \phi(\mathbf{h}_j, \mathbf{e}_{ij}). \tag{6.3}$$

Residual gated GCNs add residual connections to gated convolutions, enabling deeper architectures. Inspired by ResNets [Koo21], this enhances gradient flow and mitigates vanishing gradients, as shown in Equation 6.4.

$$\mathbf{h}_i^{(l+1)} = \mathbf{h}_i^{(l)} + \text{GatedConv}(\mathbf{h}_i^{(l)}, \{\mathbf{h}_j^{(l)}\}, \{\mathbf{e}_{ij}\}). \tag{6.4}$$

To predict the edges and nodes of the RSMTs, we also need adaptive attention mechanisms. Adaptive attention mechanisms such as those used in Graph Attention Network (GAT) [VCC+17], Graph Attention Network Version 2 (GATv2) [BAY21], allow GNNs to dynamically learn which neighbors are more important when aggregating information. This is in contrast to traditional GCNs, which use fixed, uniform, or heuristic-based aggregation (e.g., mean or sum).

Adaptive attention lets the model assign context-sensitive, data-driven weights to neighbors, which is important for node type assignment in RSMT construction. Residual gated GCNs capture structured, edge-level dependencies with learnable gating, while residual GATv2 enhances node representations through feature-driven, adaptive attention. A hybrid of the two is ideal for RSMT construction.

Table 6.1 gives the list of symbols used in this chapter and their meanings.

Table 6.1: List of Symbols and Their Meanings

Symbols	**Meaning**
K	Number of concurrent nets in a group
n	Total number of nets
$\mathcal{N} = \{N_1, N_2, \ldots, N_n\}$	Set of nets
$N_i = \{\mathrm{P}_{i1}, \mathrm{P}_{i2}, \ldots, \mathrm{P}_{i\lvert N_i\rvert}\}$	i^{th} net
P_{ij}	2D coordinate of j^{th} pin of net N_i
$\mathbf{h}_v$	Feature vector of node v

Continued on next page

Table 6.1 continued from previous page

Symbols	**Meaning**
$\mathbf{e}_{ij}$	Feature vector of edge (i, j)
$\hat{A}$	Adjacency matrix
$\hat{D}$	Degree matrix
$\mathbf{H}^{(l)}$	Node feature matrix at layer l
$\mathbf{W}^{(l)}$	Learnable weight matrix
σ	Sigmoid non-linear activation function
$\mathbf{g}_{ij}$	Gate of edge (i, j)
$\mathbf{m}_i$	Message at node i
G_j	j^{th} group
$\mathbf{fv}_i$	Feature vector of Net N_i during net grouping
λ_1, λ_2	Principal component variance ratios
C_i	i^{th} Cluster during net grouping
$p_{\max}$	Maximum number of pins per net
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Combined Hanan grid graph
$l_1 \times l_2$	Dimension of combined Hanan grid graph
$\mathbf{z}_i$	Initial feature vector of node i in combined Hanan grid
d_v	Dimension of Initial feature vector of node
d_e	Dimension of Initial feature vector of edge
e_{ij}	Initial feature vector of edge (i, j) in combined Hanan grid
t_i	Total pin count of a group G_i
$\mathcal{G}_{\text{RSMTs}} = (\mathcal{V}, \mathcal{E}_{\text{RSMTs}})$	Edge-overlap-minimized combined RSMT
$z_{c,i,j}$	Indicator variable representing the selection of a path j for a connection c in net N_i
$\{z^*_{c,i,j}\}$	Optimal solution
$x_{i,e}$	Indicator variable representing whether net N_i uses grid edge e
y_e	Total number of nets using edge e.
$\mathcal{P}_{c,i,0}$	Primary path for connection c in net N_i
$\mathcal{P}_{c,i,1}$	Alternate path for connection c in net N_i

Continued on next page

Table 6.1 continued from previous page

Symbols	Meaning
E^*	Union of all candidate edges from all nets and their path options
E^{opt}	Final set of optimized edges
$\hat{y}_{ij}^{(\text{edge})}$	Edge existence probability of edge (i, j)
$\hat{y}_i^{(\text{con})}$	Connecting node probability of node i
$\hat{y}_i^{(\text{stp})}$	Steiner point probability of node i
d_h	Hidden dimension
$\mathbf{D}_i$	Directional Distance Encoding vector of node i
$\mathcal{T}^{(k)}$	Subset of *terminal nodes* of k^{th} net in a combined Hanan grid graph.
L	Number of message passing layers in GNN model
$\mathcal{I}$	Multi-Hop interval
$\mathbf{d}_i^{(\cdot)}$	Encodes how far one must travel along a cardinal direction
$\tilde{\mathbf{z}}_i$	Updated feature vector of node i
d_{DDE}	Dimension of directional distance encoding vector
$\mathbf{W}_v$, $\mathbf{W}_h$, $\mathbf{W}_f$, $\mathbf{W}_g$, $\mathbf{W}_\alpha$, $\mathbf{W}_p$	Learnable weight matrices
$\mathbf{f}^{(l)}$	Hidden representation of edge at message passing layer l
$\mathbf{h}^{(l)}$	Hidden representation of node at message passing layer l
$c_{ij}^{(\psi)}$	Normalization constant in Multi-hop layer
$\mathbf{s}$	Virtual slot
α_{ij}	Attention score between node i and edge (i, j)
$\bar{\mathbf{v}}_i$	Mean edge descriptors
$\mathbf{b}_q, \mathbf{b}_k, \mathbf{b}_v$	Learnable biases
ζ_i	Shared node feature is built using the final JK-projected, node slot context, node edge summary, and mean edge descriptors
$\mathcal{FL}$	Focal loss

Continued on next page

Table 6.1 continued from previous page

Symbols	Meaning
$\mathcal{L}_{\text{BCE}}$	Binary cross-entropy with logits
p_t	Model-estimated probability of the true class
γ	Focusing parameter in focal loss
δ	Class-balance factor
$\mathcal{L}_{\text{edge}}, \mathcal{L}_{\text{con}}, \mathcal{L}_{\text{stp}}$, $\mathcal{L}_{\text{wl}}, \mathcal{L}_{\text{edge}}, \mathcal{L}_{\text{ctx}}, \mathcal{L}_{\text{pe}}, \mathcal{L}_{\text{total}}$	Losses for edge, connecting node, Steiner point, wirelength, context loss, positional encoding loss and total loss respectively
$\lambda_{edge}, \lambda_{con}, \lambda_{stp}, \ \lambda_{wl}, \lambda_{ctx}$ and λ_{pe}	Weights associated with losses for edge, connecting node, Steiner point, wirelength, context loss, positional encoding loss and total loss respectively
ℓ_{ij}	Length of edge (i, j)
$\hat{\mathbf{Y}}^{(\text{edge-logits})}, \hat{\mathbf{Y}}^{(\text{con-logits})}$, $\hat{\mathbf{Y}}^{(\text{stp-logits})}$	Predicted logits matrices for edge, connecting node, and Steiner point
$\hat{\mathbf{Y}}^{(\text{edge})}, \hat{\mathbf{Y}}^{(\text{con})}, \hat{\mathbf{Y}}^{(\text{stp})}$	Prediction probability matrices for edge, connecting node, and Steiner point.
$\mathcal{B}$	Branch group
$\pi_{ij}^{(N_k)}$	Edge priority score
$\texttt{NEB}(i)$	Neighbors of node i
$\hat{\mathcal{G}}^{(N_k)} = (\mathcal{V}, \hat{\mathcal{E}}^{(N_k)})$.	Predicted RSMT

6.3.2 Problem Formulation

The problem of constructing RSMTs of multiple nets concurrently can be stated as:

Given the coordinates of the pins of n nets, design a hybrid GNN model to construct RSMTs of $K \leq n$ nets concurrently that minimizes the overall time for construction, edge overlapping among the RSMTs for the K nets, and their total wirelength.

6.4 Overview of Proposed Methodology

This section presents an overview of our proposed method, covering both the dataset generation process and the model architecture. Figure 6.2 provides an outline of our proposed method. Figure 6.2(a) shows the generation of training and validation datasets, and the summary of the training of our hybrid GNN model. To form the training and validation datasets required for supervised learning, we first generate various degrees of nets with randomly generated pin coordinates. We consider the valid assumption that the pins of multiple nets of a circuit cannot have the same coordinates. During the random generation of nets, we maintain this assumption.

Next, we group nets based on pin coordinates, as briefly described in Section 6.5.1. This ensures balanced pin density and net topology within each group, avoiding both overly sparse and overly dense arrangements. We combine the pins of the nets of each group together to form a group-wise combined Hanan grid graph that serves as the input graph to our hybrid GNN model. To maintain the identity of each net, the nodes that represent the pins of a particular net are labeled with appropriate indicators.

To generate the ground truth, we first find the optimal RSMTs of individual nets of each group using `GeoSteiner` [JWWZ18], followed by an ILP-based group-wise edge overlap minimization. After that, we find the group-wise edge-overlap-minimized combined RSMTs. Like the combined Hanan grid graph, the nodes and edges of combined RSMTs are also labeled with appropriate indicator variables to maintain the identity of each net. For the combined RSMT, the nodes are labeled with two different types: pins and connecting nodes. The connecting nodes include Steiner points and the intermediate nodes that are present on an RSMT branch. The construction method of the combined Hanan grid graph and combined RSMT is described in Section 6.5.2 and Section 6.5.3, respectively.

Figure 6.2(a) illustrates the architecture of our hybrid GNN model, which includes node and edge encoding layers, residual gated GCN (ResGateGCN) [BL17] layers, residual GATv2 (ResGATv2) [BAY21] layers, and multi-hop attention [FCL+22] layers. To extract the global context from the combined Hanan grid graph, the model incorporates jumping knowledge (JK) [XLT+18], global multi-head attention

(MHA) [VSP+17], virtual slots [HLG+19], Feature-wise Linear Modulation (FiLM) [PSDV+18], and an edge attention mechanism. It employs an edge classifier to predict which edges belong to which RSMT, a node classifier to identify useful connecting nodes (pins, intermediate nodes, or Steiner points), and another node classifier to detect Steiner points. These classifiers also assign nodes to their corresponding nets. A detailed description of the architecture is provided in Section 6.6.

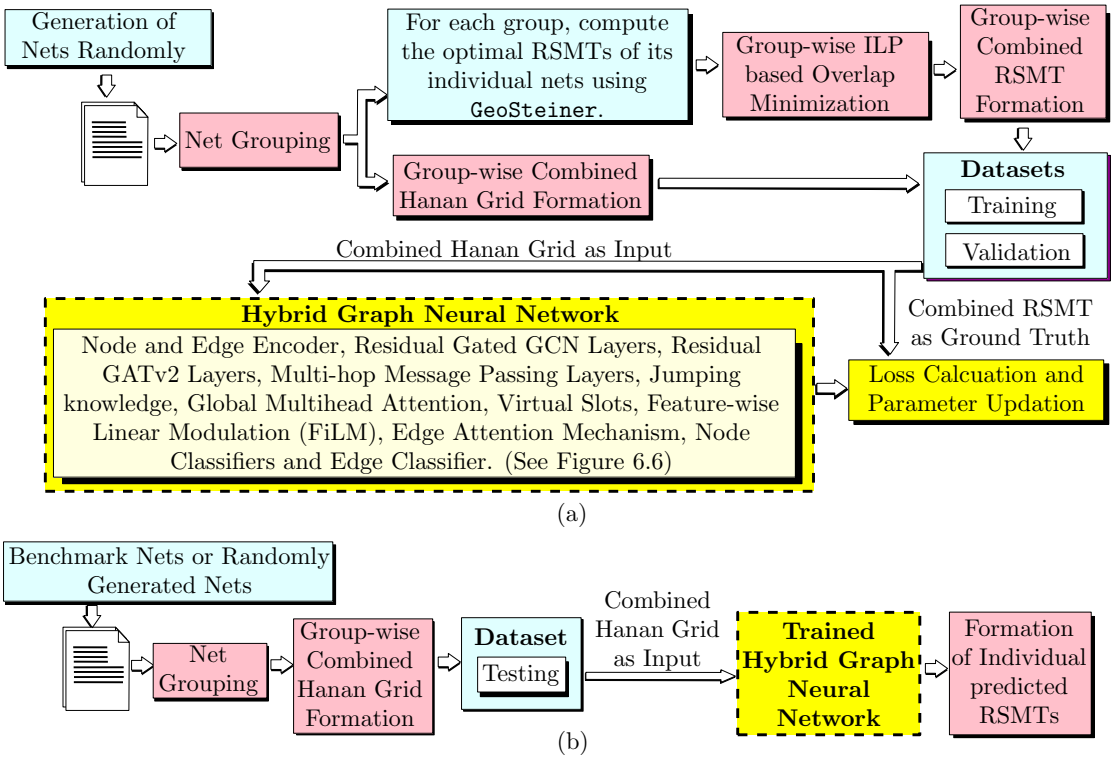


Figure 6.2: Outline of the proposed method: (a) Generation of training and validation datasets with training of our hybrid GNN model (b) Test dataset generation and RSMT prediction. Pink blocks represent our novel approaches for concurrent RSMT construction of multiple nets, yellow blocks indicate our application-specific contributions in model design and training, and cyan blocks denote general methods.

Figure 6.2(b) shows the generation of the test dataset along with the overview of the testing of our hybrid GNN model. For testing, we either need to generate random

nets or consider nets from industry benchmarks. After that, we group these nets on the basis of the pin coordinates using the same method used to generate the training and validation datasets. For each group, we generate a combined Hanan grid graph that forms our testing dataset and acts as the input to our trained hybrid GNN model. The edge classifier generates a vector that indicates edge assignments to nets, and the node classifier generates a vector to indicate a net and node type. Post-processing is required to generate individual RSMTs from these predicted indicator vectors as explained in Section 6.7.

6.5 Generation of Dataset

In this section, we describe the data generation process for training, validation, and testing.

6.5.1 Net Grouping with Fixed Group Size

In this step, we aimed to partition a set of nets into fixed-size groups of K nets, while ensuring similar topology and spatial layout across groups with balanced pin count. The input is a set of nets $\mathcal{N} = \{N_1, N_2, \ldots, N_n\}$, where $N_i = \{\mathrm{P}_{i1}, \mathrm{P}_{i2}, \ldots, \mathrm{P}_{i|N_i|}\}$ such that P_{ij} represents the 2D coordinate of j^{th} pin or terminal of net N_i. The goal is to form groups $G_1, \ldots, G_{n/K} \subset \mathcal{N}$ of size $|G_j| = K$, with similar topology and spatial layout, while keeping total pin counts across groups as uniform as possible. Both fixed group size and topology similarity are essential for correct encoding and effective training.

$$\mathbf{fv}_i = \Big[|N_i|, \mathrm{mean}(d_i), \mathrm{std}(d_i), \max(d_i),\ \lambda_1,\ \lambda_2\Big] \in \mathbb{N} \times \mathbb{R}^5 \tag{6.5}$$

Algorithm 6.1 summarizes the method. For each net N_i, we construct a feature vector $\mathbf{fv}_i \in \mathbb{N} \times \mathbb{R}^5$ (Equation 6.5). The first four are handcrafted features: the net degree $|N_i|$, the mean, standard deviation, and maximum L_1 distances ($\mathrm{mean}(d_i)$, $\mathrm{std}(d_i)$, $\max(d_i)$) between the centroid of N_i and its pins P_{ij}. The last two are learned features, λ_1 and λ_2, denoting the principal component variance ratios obtained from

principal component analysis (PCA) on N_i, capturing its topological and spatial layout.

Algorithm 6.1: Net Grouping with Fixed Group Size

Input : Netlist $\mathcal{N} = \{N_1, \ldots, N_n\}$, where each net N_i has pins $\mathbf{P}_{ij} \in \mathbb{R}^2$; group size K

Output: Groups $\{G_1, \ldots, G_{\lceil n/K \rceil}\}$ with similar topology and spatial layout, balanced by pin count, each of size K

foreach $N_i \in \mathcal{N}$ **do**
  Compute centroid $\boldsymbol{\mu}_i = \frac{1}{|N_i|} \sum_{j=1}^{|N_i|} \mathbf{P}_{ij}$;
  Compute distances $d_i = \{\|\mathbf{P}_{ij} - \boldsymbol{\mu}_i\|_1 \mid j = 1, \ldots, |N_i|\}$;
  Compute mean(d_i), std(d_i), and max(d_i);
  Compute PCA eigenvalues $\lambda_1, \lambda_2 = \text{PCA}(N_i)$;
  Construct feature vector $\mathbf{fv}_i = [|N_i|,\ \text{mean}(d_i),\ \text{std}(d_i),\ \max(d_i),\ \lambda_1,\ \lambda_2]$;
Standardize feature matrix $\mathbf{FM} = \{\mathbf{fv}_i \mid N_i \in \mathcal{N}\}$;
$C_1, C_2, \ldots, C_K \leftarrow \text{GreedyBalancedCluster}(\mathbf{FM}, K)$;
Create empty groups $G_1, \ldots, G_{\lceil n/K \rceil}$;
Initialize pin counters $p_j \leftarrow 0$ for all j;
foreach *cluster* $C_k \in \{C_1, \ldots, C_K\}$ **do**
  if $|C_k| < \frac{n}{K}$ **then**
    Extend C_k by duplicating elements;
  Sort nets in C_k in non-ascending order of degree $|N_i|$;
  Sort groups $\{G_j\}$ in non-descending order of total pins p_j;
  foreach *net* N_i *in sorted* C_k **do**
    Assign N_i to next group G_j in sorted order;
    Update $p_j \leftarrow p_j + |N_i|$;
return $\{G_1, G_2, \ldots, G_{\lceil n/K \rceil}\}$;

The feature vectors are assembled into a feature matrix **FM** and normalized to ensure all features are on the same scale. **FM** is then clustered into K equal-

Algorithm 6.2: GreedyBalancedCluster

Input : Feature matrix **FM**; number of clusters K
Output: Clusters $\{C_1, C_2, \ldots, C_K\}$ with $|C_i| = \lceil n/K \rceil$

Run K-Means on **FM** to obtain centroids $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K\}$;
Compute $D_{ij} = \|\mathbf{FM}_i - \boldsymbol{\mu}_j\|_2$ for all $i \in [1, n], j \in [1, K]$;
Initialize empty clusters $C_1, \ldots, C_K$;
Initialize assignment array $a[i] \leftarrow -1$ for all $i \in [1, n]$;
Form list $L = \{(i, j, D_{ij}) \mid i \in [1, n],\ j \in [1, K]\}$;
Sort L in non-decreasing order of D_{ij};
foreach $(i, j, D_{ij}) \in L$ **do**
    if $a[i] = -1$ ***and*** $|C_j| < \lceil n/K \rceil$ **then**
        Assign net N_i to cluster C_j and set $a[i] \leftarrow j$;
foreach i *such that* $a[i] = -1$ **do**
    $j^* \leftarrow \arg\min_j |C_j|$;
    Assign net N_i to cluster C_{j^*} and set $a[i] \leftarrow j^*$;
return $\{C_1, C_2, \ldots, C_K\}$;

sized clusters $C_1, C_2, \ldots, C_K$ using a greedy balanced clustering algorithm described in Algorithm 6.2 with cluster size $\lceil \frac{n}{K} \rceil$.

Next, the method constructs new $\lceil \frac{n}{K} \rceil$ groups of size K from $C_1, C_2, \ldots, C_K$ clusters while balancing the total number of pins in each group. First, $\lceil \frac{n}{K} \rceil$ number of empty groups are initialized, and their total pin counts are set to zero. For each original cluster, if it contains fewer nets than required, its elements are duplicated to ensure enough nets are available for assignment. Within each cluster, nets are prioritized by size (degree), with larger nets considered first to minimize imbalance. Simultaneously, groups are ordered by their current total pin counts, with the least-loaded groups receiving the next net from the cluster. This process is repeated iteratively for all clusters, assigning one net from each cluster to each group in a manner that greedily balances the total pin counts. As a result, every group ends up with exactly one net from each cluster, and the variation in total pins across groups is minimized.

The overall time complexity of the method is dominated by three main stages: feature extraction, clustering, and group balancing. Feature extraction takes $O(n \cdot p_{\max})$, where $p_{\max}$ denotes maximum count of pins per net. Standardization of the

feature matrix is $O(n)$ and the greedy balanced clustering in Algorithm 6.2 takes $O(n \cdot K \cdot \log(n \cdot K))$. The greedy assignment of nets to groups involves sorting nets within each cluster and sorting groups by current pin counts, contributing $O(n \log n)$ overall. Hence, the total time complexity is $O(n \cdot p_{\max} + n \cdot K \cdot log(n \cdot K) + n \log n)$.

6.5.2 Combined Hanan Grid Graph Construction

To construct the combined Hanan grid graph for a group $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes $\mathcal{V}$, edges $\mathcal{E}$, we begin by collecting all pins from every net within the group. We then extract the unique $x-$ and $y-$ coordinates from these pins and sort them in ascending order. Let the Hanan grid size be $l_1 \times l_2$ where l_1 and l_2 denote the number of unique $x-$ and $y-$ coordinates.

Each node $i \in \mathcal{V}$ is represented by an initial feature vector $\mathbf{z}_i \in \mathbb{R}^{d_v}$, which encodes its spatial coordinates and node type. Since each group contains K nets, the feature dimension is $d_v = 2 + K$. The first two entries of $\mathbf{z}_i$ store the node's x- and y-coordinates, while the remaining K entries are binary indicators corresponding to the K member nets. Each indicator specifies whether the node serves as a pin of the respective net. Each edge $(i, j) \in \mathcal{E}$ is associated with a feature vector $\mathbf{e}_{ij} \in \mathbb{R}^{d_e}$ such that $d_e = 1 + K$. The first field of e_{ij} stores the L_1 distance between the two incident nodes i and $j \in \mathcal{V}$, while the remaining K fields are binary indicators denoting membership of the Hanan edge in each of the K nets.

To illustrate, consider an example of a group containing 3 nets. The pin coordinates are shown in Table 6.2, Table 6.3 and Table 6.4. Figure 6.3 shows the combined Hanan grid graph with node and edge feature vectors. First, we combine all pins of the three given nets. Next, we find the unique coordinates. In this example, we have 24 unique $x-$ and 24 unique $y-$ coordinates. Hence, the combined Hanan grid graph size is 24×24.

Table 6.2: Pin coordinates corresponding to a 10-degree net N_1

N_1	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}	P_{18}	P_{19}
x	73	8	91	42	55	29	60	4	83	16
y	21	65	43	88	12	76	30	99	57	3

Table 6.3: Pin coordinates corresponding to a 10-degree net N_2

N_2	P_{20}	P_{21}	P_{22}	P_{23}	P_{24}	P_{25}	P_{26}	P_{27}	P_{28}	P_{29}
x	12	67	45	89	34	76	5	98	27	53
y	94	23	78	11	56	88	39	2	61	70

Table 6.4: Pin coordinates corresponding to a 6-degree net N_3

N_3	P_{30}	P_{31}	P_{32}	P_{33}	P_{34}	P_{35}
x	53	15	78	6	91	37
y	87	63	29	94	10	51

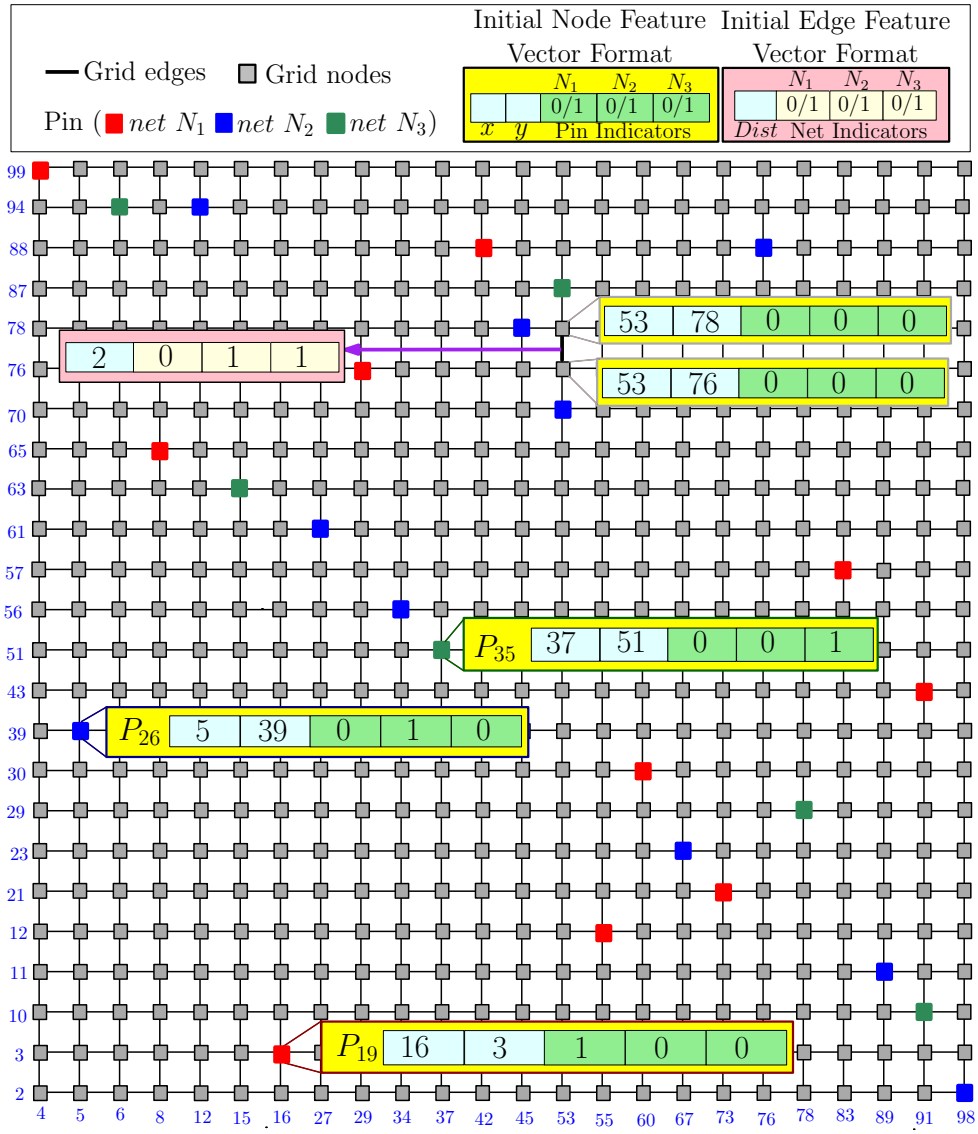


Figure 6.3: Combined Hanan grid graph with node and edge feature vectors

In Figure 6.3, red, blue and green colors indicate net N_1, N_2 and N_3 respectively. As the group contains three nets, i.e., $K = 3$, $d_v = (2+K) = 5$ and $d_e = (1+K) = 4$. Figure 6.3 shows examples of node and edge feature vectors. In an edge feature vector, multiple net indicators may be set to 1, reflecting cases where a single Hanan edge is shared by multiple nets.

Let us define the total pin count of a group G_i as $t_i = \sum_{N_j \in G_i} |N_j|$. The time complexity to perform union and sorting all pins across the nets is $O(t_i \log t_i)$. Computing node feature vectors for all nodes requires $O(K \cdot l_1 \cdot l_2)$ time, where $l_1 \times l_2$ is the combined Hanan grid dimension. Finding initial feature vectors of all edges also takes $O(K \cdot l_1 \cdot l_2)$ time as there are $(l_1 - 1) \cdot l_2$ horizontal and $(l_2 - 1) \cdot l_1$ vertical edges. Hence, the overall time complexity is $O(t_i \log t_i + K \cdot l_1 \cdot l_2)$.

6.5.3 Edge-Overlap-Minimized Combined RSMT Construction

To construct the edge-overlap-minimized combined RSMT for a group $\mathcal{G}_{\text{RSMTs}} = (\mathcal{V}, \mathcal{E}_{\text{RSMTs}})$, we took the corresponding combined Hanan grid graph and the optimal RSMTs of individual nets obtained from `GeoSteiner` [JWWZ18], as input. At first, we split each optimal RSMT edge depending on the Hanan grid graph edges.

For example, if an RSMT edge exists between two nodes $v_1(x_1, y_1)$ and $v_2(x_2, y_1)$ and an intermediate node $v_3(x_3, y_1)$ exists such that $x_1 < x_3 < x_2$ then we split the edge into two edges $v_1(x_1, y_1)$ to $v_3(x_3, y_1)$ and $v_3(x_3, y_1)$ to $v_2(x_2, y_1)$. In other words, we split the RSMT edges that span multiple Hanan grid edges. Next, we minimize the edge overlap of different RSMTs present in each group, followed by the combined RSMT construction.

6.5.3.1 Group-wise Edge Overlap Minimization

For concurrent RSMT construction of multiple nets, their RSMTs may share common grid edges. Such edge overlaps can lead to routing congestion and degrade performance. To mitigate this, we introduce a group-wise global optimization procedure that jointly refines the RSMTs of all concurrent nets. The goal is to minimize inter-net overlaps while maintaining connectivity and without wirelength increase.

Let there be K nets represented as $\mathcal{N}' = \{N_1, N_2, \ldots, N_K\}$ in a combined Hanan

Table 6.5: Decision variables used in the ILP formulation.

Symbol	Type	Meaning
$z_{c,i,j}$	binary	1 if path j ($j \in \{0,1\}$) is chosen for connection c in net N_i or if only one path ($j = 0$) exists; 0 otherwise.
$x_{i,e}$	binary	1 if net N_i uses grid edge e; 0 otherwise.
y_e	integer	Total number of nets using edge e.

grid graph. The RSMT of each net N_i consists of several connecting paths between pin-to-pin, pin-to-Steiner or Steiner-to-Steiner in the combined RSMT. For every such connecting path c in the RSMT of net N_i, we define up to two L_1 path options on the grid: $\mathcal{P}_{c,i,0}$: primary path, $\mathcal{P}_{c,i,1}$: alternate path (if available). The optimization jointly chooses one path per connection across all nets such that the total number of overlapping edges is minimized. The Integer Linear Programming (ILP) formulation is presented in Figure 6.4, with details of the decision variables provided in Table 6.5.

$$\min_{z_{c,i,k}} \sum_{e \in E^*} (y_e - 1) \tag{6.6}$$

subject to:

$$\sum_{j \in \{0,1\}} z_{c,i,j} = 1 \qquad \forall\, c,\ i, \tag{6.7}$$

$$x_{i,e} = \sum_{j \in \{0,1\}:\, e \in \mathcal{P}_{c,i,j}} z_{c,i,j} \qquad \forall\, c,\ e \in E^*, \tag{6.8}$$

$$y_e = \sum_{i=1}^{K} x_{i,e} \qquad \forall\, e \in E^*, \tag{6.9}$$

$$z_{c,i,j} \in \{0,1\}, \quad x_{i,e} \in \{0,1\}, \quad y_e \geq 0. \tag{6.10}$$

Figure 6.4: Generalized ILP for edge overlap minimization of combined RSMTs in each group

Let E^* denote the union of all candidate edges from all nets and their path options. The objective in Equation 6.6 minimizes the total overlap across all edges, where $(y_e - 1)$ measures the excess usage beyond one net, implying that overlap occurs only if multiple nets share the same edge. The path selection constraint in Equation 6.7 ensures that for each connection, exactly one path is chosen. The edge

activation constraint in Equation 6.8 marks edge e as used by net N_i if it lies on any of the selected candidate paths for that net. The edge usage counting constraint in Equation 6.9 aggregates how many nets use edge e, i.e., the total demand on that edge. Finally, the variable domain constraint in Equation 6.10 enforces binary or non-negative values for all decision variables.

The optimal solution $\{z^*_{c,i,j}\}$ determines which path each connection should take. The final set of optimized edges is obtained as shown in Equation 6.11. This optimized configuration updates the edge and node labels in the combined RSMT graph, achieving a global balance between minimizing overlaps and preserving wirelength across all concurrent nets.

$$E^{\text{opt}} = \bigcup_{c,i,j:\, z^*_{c,i,j}=1} \mathcal{P}_{c,i,j} \tag{6.11}$$

6.5.3.2 Combined RSMT Construction

We have defined four node types for an RSMT as follows:

- Node representing pin
- Node representing Steiner point
- Connecting intermediate nodes that are produced due to the RSMT edge splitting
- Unused nodes.

If we have K nets in a group, then the size of the node label is $3K + 2$. The first two fields of the node label contain the $x-$ and $y-$ coordinates. The next K fields are the net-wise indicator of whether the node corresponds to a pin of any net or not. The next K fields are the net-wise indicator of whether the node corresponds to a Steiner point of any net or not, and the last K fields are the net-wise indicator of whether the node corresponds to a connecting node or not.

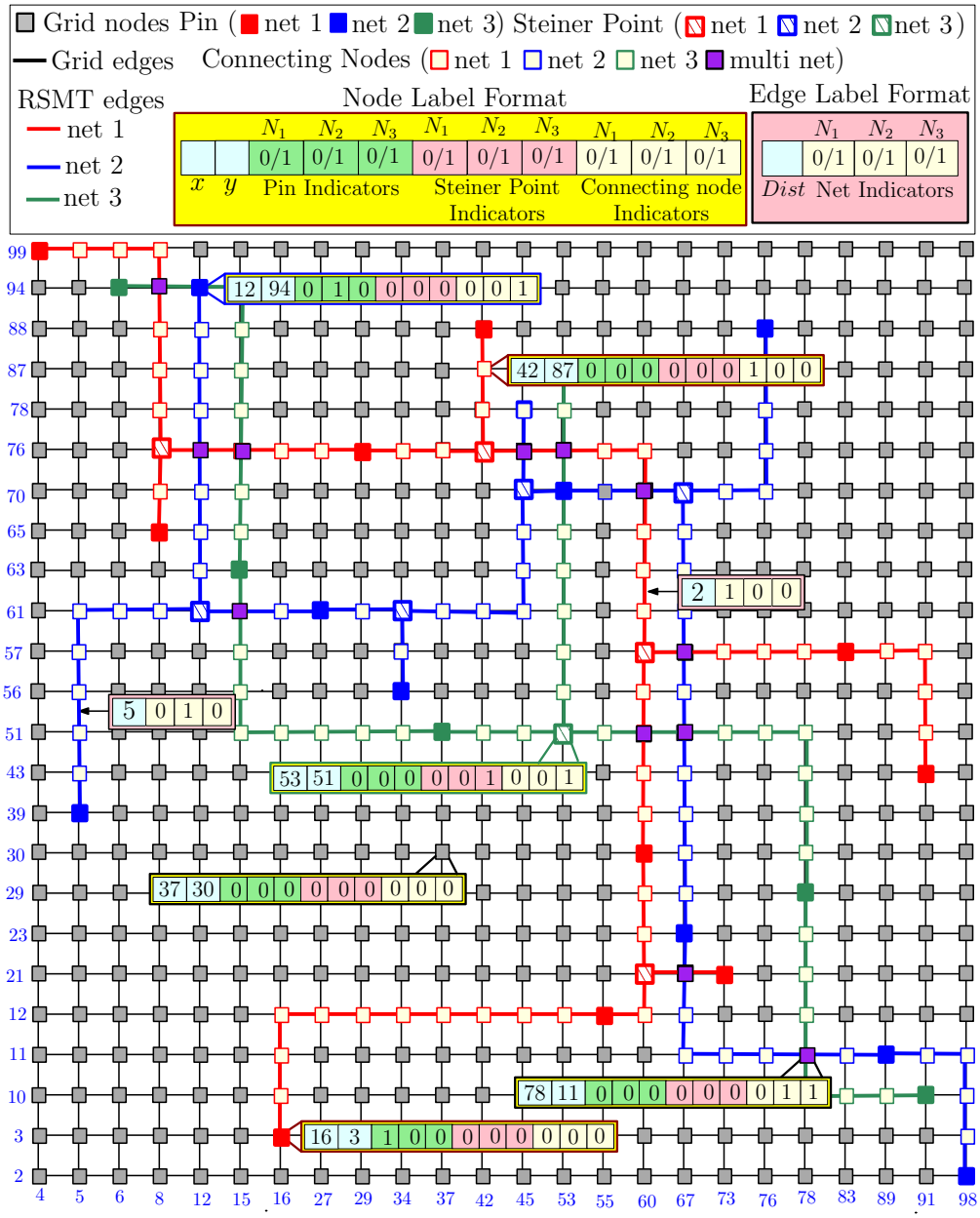


Figure 6.5: Combined RSMT with node and edge labels

The size of the edge label is $K + 1$, where the first field stores the L_1 distance between the nodes with which the edge is connected. The next K fields are the net-

wise indicator to specify whether the edge belongs to any RSMT or not. If the edge belongs to only one RSMT, then the corresponding field will be 1. If the edge belongs to multiple RSMTs, then all the corresponding net indicators will contain 1.

Figure 6.5 shows the combined RSMT of the example combined Hanan grid graph shown in Figure 6.3 and Table 6.2, Table 6.3, and Table 6.4. In the example, since we have three nets, the size of the node label is 11 and the size of the edge label is 4. Figure 6.5 shows the node and edge labels of a few nodes and edges. For unused nodes, all the indicator fields will contain 0.

The time complexity of splitting all RSMT edges and labeling nodes and edges is $O(K \cdot l_1 \cdot l_2)$. In contrast, the group-wise ILP-based edge overlap minimization across multiple RSMTs has exponential complexity. Nonetheless, this cost is acceptable as it is incurred only once during training and validation dataset preparation, not during testing.

6.6 Our Hybrid GNN Model Architecture and Loss

Our hybrid GNN model is designed for joint node and edge prediction, and its complete architecture is illustrated in Figure 6.6. The model begins with two linear encoders that project the raw node and edge features into a shared hidden dimension. It then applies a combination of Residual Gated GCN (ResGateGCN) [BL17] layers, Residual GATv2 (ResGATv2) [BAY21] layers, and a Multi-Hop [FCL+22] message passing layer, combining gated message passing, attention-based aggregation, and long-range context to capture both structural and feature-dependent relationships. To capture global context, Jumping knowledge [XLT+18], Global Multi-head Attention [VSP+17], Virtual slots [HLG+19], and Feature-wise Linear Modulation (FiLM) [PSDV+18] are included in our model. For edge prediction, an edge attention mechanism computes relevance scores to inform decoding.

The connecting node prediction head employs a multilayer perceptron (MLP) to classify whether a node serves as a useful connecting node within a given net, while the Steiner point prediction head utilizes another MLP to determine whether a node functions as a Steiner point of that net. The edge prediction module consists of multiple components: a local edge decoder (a sequential MLP) that predicts edge

existence from neighboring node and edge features, a global edge decoder (another MLP) that incorporates global context, a gated mechanism (MLP with LayerNorm and sigmoid) that dynamically weights or filters edge connections, and a final linear layer that produces the edge prediction logits.

This architecture is particularly well-suited for learning structured relationships in Hanan grid graphs. The gating and attention mechanism allows the model to dynamically control which messages are most important, improving expressiveness over standard GCNs. The residual connections and layer normalization improve training depth and stability, enabling the model to learn richer hierarchical representations. The model aims to jointly predict:

- Edge existence probabilities $\hat{y}_{ij}^{(\text{edge})}\ \forall (i,j) \in \mathcal{E}$.
- Connecting node probabilities $\hat{y}_{i}^{(\text{con})}\ \forall i \in \mathcal{V}$.
- Steiner point probabilities $\hat{y}_{i}^{(\text{stp})}\ \forall i \in \mathcal{V}$

6.6.1 Input Encoding using Linear Encoders

In our hybrid GNN framework, the node and edge features are first embedded into a shared hidden space of dimension d_h, with distinct transformations applied to each to capture their respective roles in the graph.

6.6.1.1 Node Feature Encoding

The node features are enhanced with Directional Distance Encoding [BPL+21] (DDE). The DDE vector $\mathbf{D}_i \in \mathbb{R}^{\text{dDDE}}$ provides spatial context relative to terminals. It encodes both *magnitude* and *orientation* of proximity to terminals, acting as a directional morphological distance transform. Let $i \in \mathcal{V}$ have cartesian coordinates $\mathbf{p}_i = (x_i, y_i) \in \mathbb{R}^2$. Let $\mathcal{T}^{(k)} \subseteq \mathcal{V}$ denote the subset of *terminal nodes* of k^{th} net in a combined Hanan grid graph. The DDE vector is defined in Equation 6.12.

$$\mathbf{D}_i = \mathbf{concat}\left(\mathbf{d}_i^{(1)}, \mathbf{d}_i^{(2)}, \ldots, \mathbf{d}_i^{(K)}\right) \in \mathbb{R}^{4K} \tag{6.12}$$

where each component $\mathbf{d}_i^{(\cdot)}$ encodes how far one must travel along a cardinal direction (left, right, up or down) from node i of the net k until the nearest terminal of

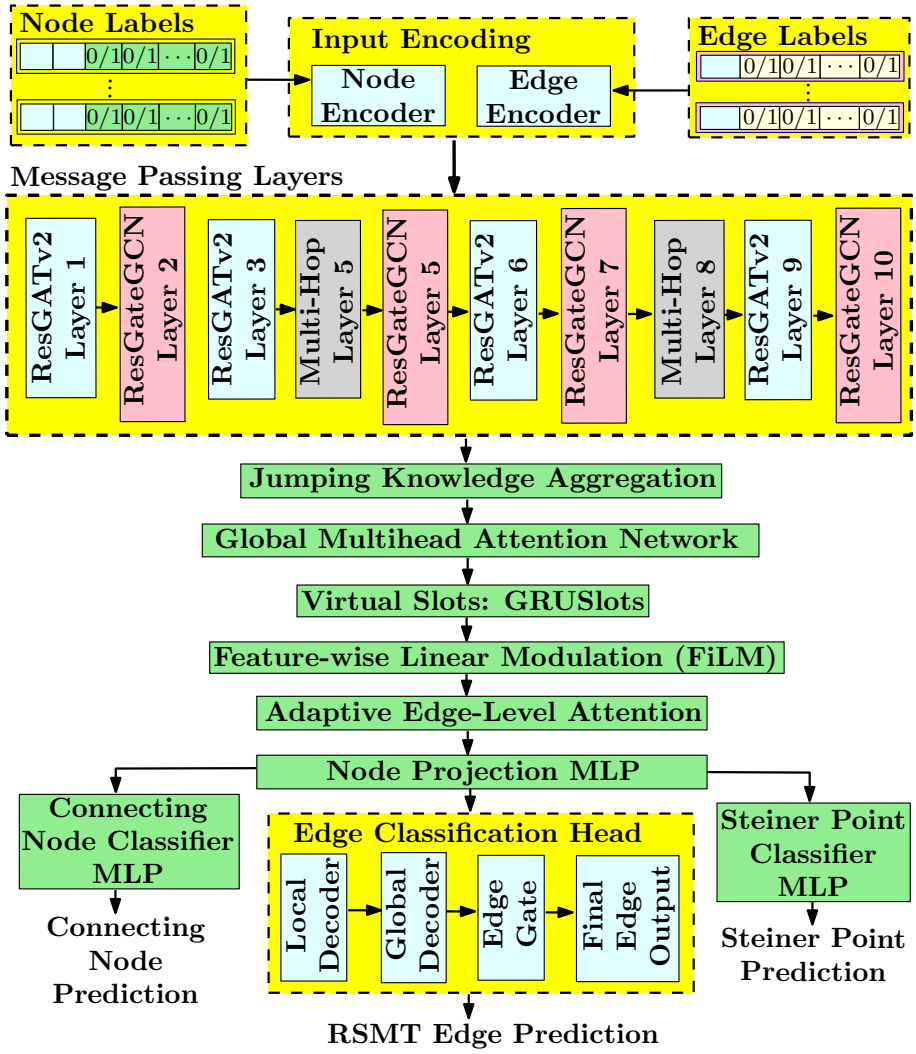


Figure 6.6: Our Hybrid GNN Model Architecture by considering the number of message passing layers $L = 10$, multi-hop interval $\mathcal{I} = 4$, and following Equation 6.17 and Equation 6.18.

net k, and is defined as $\mathbf{d}_i^{(k)} = \mathbf{concat}\left(\mathbf{d}_i^{(k),L}, \mathbf{d}_i^{(k),R}, \mathbf{d}_i^{(k),U}, \mathbf{d}_i^{(k),D}\right)$ for all $k = 1$ *to* K. If no terminal exists in a direction, the algorithm substitutes the distance to the farthest node in the same direction, ensuring that all directional distances are finite and continuous across the spatial domain. All components of $\mathbf{d}_i^{(k)}$ are defined in Equation 6.13 - Equation 6.16.

$$\mathbf{d}_i^{(k),L} = \begin{cases} \min\limits_{j \in \mathcal{T}^{(k)}, x_j < x_i} (x_i - x_j), & \text{if such } j \text{ exists,} \\ x_i - \min\limits_{j : x_j < x_i} x_j, & \text{otherwise,} \end{cases} \tag{6.13}$$

$$\mathbf{d}_i^{(k),R} = \begin{cases} \min\limits_{j \in \mathcal{T}^{(k)}, x_j > x_i} (x_j - x_i), & \text{if such } j \text{ exists,} \\ \max\limits_{j : x_j > x_i} x_j - x_i, & \text{otherwise,} \end{cases} \tag{6.14}$$

$$\mathbf{d}_i^{(k),U} = \begin{cases} \min\limits_{j \in \mathcal{T}^{(k)}, y_j < y_i} (y_i - y_j), & \text{if such } j \text{ exists,} \\ y_i - \min\limits_{j : y_j < y_i} y_j, & \text{otherwise,} \end{cases} \tag{6.15}$$

$$\mathbf{d}_i^{(k),D} = \begin{cases} \min\limits_{j \in \mathcal{T}^{(k)}, y_j > y_i} (y_j - y_i), & \text{if such } j \text{ exists,} \\ \max\limits_{j : y_j > y_i} y_j - y_i, & \text{otherwise.} \end{cases} \tag{6.16}$$

The updated node feature vector: $\tilde{\mathbf{z}}_i = \mathbf{concat}(\mathbf{z}_i, \mathbf{D}_i) \in \mathbb{R}^{d_v + d_{\text{DDE}}}$. To project this input into a common latent space suitable for graph message passing, a linear transformation is applied as $\mathbf{h}_i^{(0)} = \mathbf{W}_v \, \tilde{\mathbf{z}}_i + \mathbf{b}_v$, where $\mathbf{W}_v \in \mathbb{R}^{d_h \times (d_v + d_{\text{DDE}})}$ is a learnable weight matrix, $\mathbf{b}_v \in \mathbb{R}^{d_h}$ is a bias vector, $\mathbf{h}_i^{(0)} \in \mathbb{R}^{d_h}$ is the encoded node embedding. This transformation maps heterogeneous node features and DDE cues into a shared hidden representation space of dimension d_h.

6.6.1.2 Edge Feature Encoding

Similarly, each edge (i, j) in the graph carries its own feature vector $\mathbf{e}_{ij} \in \mathbb{R}^{d_e}$. These are projected directly using $\mathbf{f}_{ij}^{(0)} = \mathbf{W}_e \mathbf{e}_{ij} + \mathbf{b}_e$, where $\mathbf{W}_e \in \mathbb{R}^{d_h \times d_e}$ and $\mathbf{b}_e \in \mathbb{R}^{d_h}$ are learnable parameters.

6.6.2 Message Passing Layers

After feature extraction, we construct a multi-layer GNN with heterogeneous architectures, integrating gated convolutions, attention, and higher-order aggregation. Intermediate representations are projected and unified via a jumping knowledge (JK) [XLT+18] mechanism.

Let L be the number of message passing layers and $\mathcal{I}$ the MultiHop [FCL+22] interval. For $l \in \{1, \ldots, L\}$, let $\mathbf{h}^{(l)}$ and $\mathbf{f}^{(l)}$, denote the hidden representation of nodes and edges at message passing layer l. The layer arrangement is shown in Equation 6.17 and the condition $C(l)$ is defined in Equation 6.18.

$$\text{Layer}(l) = \begin{cases} \text{ResGateGCN}\big(\mathbf{h}^{(l-1)}, \mathbf{f}^{(l-1)}\big) & C(l) = 2 \\ \text{ResGATv2}\big(\mathbf{h}^{(l-1)}, \mathbf{f}^{(l-1)}\big) & C(l) = 1 \\ \text{MultiHop}\big(\mathbf{h}^{(l-1)}, \Psi\big) & C(l) = 0 \end{cases} \tag{6.17}$$

$$C(l) = \begin{cases} 0, & l \bmod \mathcal{I} = 0 \\ 1 + \big((l - \left\lfloor \frac{l-1}{\mathcal{I}} \right\rfloor - 1) \bmod 2\big), & \text{otherwise.} \end{cases} \tag{6.18}$$

where,

- ResGateGCN [BL17] is a Residual Gated Graph Convolutional layer that models local structural information with gating mechanisms as described in Section 6.6.2.1.
- ResGATv2 [BAY21] is a residual Graph Attention v2 convolutional layer that uses attention to weight neighbors dynamically as described in Section 6.6.2.2.
- MultiHop [FCL+22] aggregates information from its $1-$ to $\Psi-$hop neighborhoods of each node, capturing high-order dependencies as described in Section 6.6.2.3.

Note that after each MultiHop layer, the order of ResGATv2 and ResGateGCN layers alternates to preserve the pattern shown in Figure 6.6. The frequency of MultiHop layers and the total number of message-passing layers are chosen such that the final layer is ResGateGCN, as it jointly updates both node and edge features.

6.6.2.1 Residual Gated Graph Convolutional Layer

The residual gated graph convolution layer jointly updates node and edge features by incorporating gated message passing, residual connections, and normalization. Let each node i be associated with an encoded feature vector $\mathbf{h}_i \in \mathbb{R}^{d_\text{h}}$ and each edge (i, j) with an encoded feature vector $\mathbf{f}_{ij} \in \mathbb{R}^{d_\text{h}}$. As a preprocessing step, these features are projected into a shared hidden space via linear transformations: $\tilde{\mathbf{h}}_i = \mathbf{W}_h \mathbf{h}_i$ and $\tilde{\mathbf{f}}_{ij} = \mathbf{W}_f \mathbf{f}_{ij}$ where $\mathbf{W}_h, \mathbf{W}_f \in \mathbb{R}^{d_h \times d_h}$ are learnable parameters.

For message passing, each neighbor $j \in \mathcal{N}(i)$ sends a gated message to node i. The gate is defined in Equation 6.19 and the message content is in Equation 6.20 where $\mathbf{W}_g, \mathbf{W}_m \in \mathbb{R}^{d_h \times d_h}$ are learnable matrices and $\odot$ denotes elementwise multiplication.

$$\mathbf{g}_{ij} = \sigma\Big(\mathbf{W}_g(\tilde{\mathbf{h}}_j + \tilde{\mathbf{f}}_{ij})\Big)\,, \tag{6.19}$$

$$\mathbf{m}_{ij} = \mathbf{g}_{ij} \odot \mathbf{W}_m(\tilde{\mathbf{h}}_j + \tilde{\mathbf{f}}_{ij}), \tag{6.20}$$

Node i aggregates its incoming messages using an element-wise mean as shown in Equation 6.21, and the node feature is then updated with residual normalization as shown in Equation 6.22, where $\mathrm{LN}(\cdot)$ represents layer normalization.

$$\mathbf{m}_i = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij}. \tag{6.21}$$

$$\mathbf{h}_i^{\text{out}} = \mathrm{ReLU}(\mathrm{LN}(\mathbf{h}_i + \mathbf{m}_i))\,. \tag{6.22}$$

In parallel, each edge (i, j) updates its embedding by including the updated node embeddings at its endpoints together with the transformed edge feature as shown in Equation 6.23, followed by a residual update with normalization as shown in Equation 6.24.

$$\mathbf{f}_{ij}^{\text{msg}} = \mathbf{h}_i^{\text{out}} + \mathbf{h}_j^{\text{out}} + \tilde{\mathbf{f}}_{ij}, \tag{6.23}$$

$$\mathbf{f}_{ij}^{\text{out}} = \mathrm{ReLU}\big(\mathrm{LN}(\mathbf{f}_{ij} + \mathbf{f}_{ij}^{\text{msg}})\big)\,. \tag{6.24}$$

This formulation enables a bidirectional flow of information: edge features influence gated node messages, while updated node embeddings refine edge attributes.

The residual connections improve gradient flow. Gating and normalization provide adaptivity and stability to our deep architecture.

6.6.2.2 Residual Graph Attention v2 Convolutional layer

The Residual Graph Attention v2 (GATv2) Convolutional layer integrates adaptive attention-based neighbor aggregation with residual connections and normalization for stable training. Each layer applies the GATv2 mechanism, which computes attention coefficients from both node and edge features. In our architecture (Figure 6.6), each residual GATv2 layer consumes the updated node and edge embeddings produced by the preceding residual gated GCN layer. For each edge (i, j), the attention coefficient is given by the Equation 6.25 where $\mathbf{W}_\alpha \in \mathbb{R}^{\tilde{d}\times(2d_h+d_h)}$ is a learnable weight matrix, $\mathbf{a} \in \mathbb{R}^{\tilde{d}}$ is the attention vector.

$$\alpha_{ij} = \text{softmax}_j\big(\mathbf{a}^\top \text{LeakyReLU}(\mathbf{W}_\alpha \cdot \mathbf{concat}\,(\mathbf{h}_i, \mathbf{h}_j, \mathbf{f}_{ij}))\big)\,, \tag{6.25}$$

The aggregated representation for node i is then computed as shown in Equation 6.26 where $\mathbf{W}_v \in \mathbb{R}^{d_h\times d_h}$ is a learnable value transformation. Multi-head attention with $H = 4$ heads is used, and their outputs are averaged.

$$\mathbf{h}_i^{\text{agg}} = \sum_{j\in\mathcal{N}(i)} \alpha_{ij}\,\mathbf{W}_v\mathbf{h}_j \quad \in \mathbb{R}^{d_h}, \tag{6.26}$$

Finally, residual learning, normalization, and non-linearity are applied as $\mathbf{h}_i^{\text{res}} = \mathbf{h}_i + \mathbf{h}_i^{\text{agg}}$, $\mathbf{h}_i^{\text{norm}} = \text{LN}(\mathbf{h}_i^{\text{res}})$ and $\mathbf{h}_i^{\text{out}} = \text{ReLU}(\mathbf{h}_i^{\text{norm}})$. This formulation enables the network to dynamically modulate attention across neighbors using both node and edge features, while residual and normalization layers stabilize feature propagation in deeper architectures.

6.6.2.3 Multi-hop Message Passing Layer

For each node $i \in \mathcal{V}$, a Ψ-hop message passing layer simultaneously aggregates information from its 1- to Ψ-hop neighborhoods. Let $\mathbf{h}_i$ denote the i^{th} node's feature vector. At hop $\psi \in \{1, \ldots, \Psi\}$, we compute using Equation 6.27 where $\texttt{NBR}^\psi(i)$ is the set

of nodes at distance exactly ψ from i, $c_{ij}^{(\psi)}$ is a normalization constant, $\mathbf{W}^{(\psi)} \in \mathbb{R}^{d_h \times d_h}$ is the learnable weight for the ψ-hop convolution, and $\sigma(\cdot)$ is a nonlinear activation function.

$$\mathbf{h}_i^{(\psi)} = \sigma\left(\sum_{j \in \texttt{NBR}^{\psi}(i)} \frac{1}{c_{ij}^{(\psi)}} \mathbf{W}^{(\psi)} \mathbf{h}_j \right) \tag{6.27}$$

Representations from all hops are concatenated in Equation 6.28. A projection operator fuses the multi-hop features into the hidden dimension where $\mathbf{W}^{\text{proj}} \in \mathbb{R}^{\Psi \cdot d_h \times d_h}$ and finally, residual normalization is applied as shown in Equation 6.29.

$$\mathbf{a}_i = \textbf{concat}\left(\mathbf{h}_i^{(1)}, \mathbf{h}_i^{(2)}, \ldots, \mathbf{h}_i^{(\Psi)} \right) \in \mathbb{R}^{\Psi \cdot d} \tag{6.28}$$

$$\tilde{\mathbf{h}}_i = \sigma\big(\mathbf{W}^{\text{proj}}\, \mathbf{a}_i\big), \quad \mathbf{h}_i^{\text{out}} = \text{Norm}\, \big(\tilde{\mathbf{h}}_i + \mathbf{h}_i^{(0)}\big). \tag{6.29}$$

6.6.3 Jumping Knowledge Aggregation

The Jumping Knowledge (JK) mechanism aggregates all intermediate representations to form a comprehensive node embedding. After each message passing layer, the output is projected to a lower-dimensional slice suitable for the jumping knowledge mechanism and the projection is defined as $\mathbf{P}^{(l)} = \mathbf{W}_p^l \mathbf{h}^{(l)}$ that maps $\mathbf{P}^{(i)} : \mathbb{R}^{d_h} \to \mathbb{R}^{d_{JK}}$, where d_{JK} is the projection dimension. By collecting $\{\tilde{\mathbf{P}}^{(l)}\}_{l=1}^{L}$, the JK mechanism can flexibly combine information across multiple depths, allowing the final representation to balance shallow and deep signals. Here, we use JK with concatenation mode followed by a projection into the hidden space. The per-node JK representation is shown in Equation 6.30, where L denotes the number of message passing layers, and $d_{\text{JK}} = d_h/2$ when no separate projection dimension is specified. The corresponding JK representation matrix is shown in Equation 6.31.

$$\mathbf{h}_i^{(\text{JK})} = \textbf{concat}\left(\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)}, \ldots, \mathbf{P}_i^{(L)} \right) \in \mathbb{R}^{L \cdot d_{\text{JK}}} \tag{6.30}$$

$$\mathbf{H}^{(\text{JK})} = \begin{bmatrix} \left(\mathbf{h}_1^{(\text{JK})}\right)^{\top} \\ \vdots \\ \left(\mathbf{h}_{|\mathcal{V}|}^{(\text{JK})}\right)^{\top} \end{bmatrix} \in \mathbb{R}^{|\mathcal{V}| \times (L \cdot d_{\text{JK}})}. \tag{6.31}$$

Finally, a linear projection is applied to map the concatenated embeddings back to the hidden dimension (Equation 6.32) where $\mathbf{W}_{\text{JK}} \in \mathbb{R}^{(L \cdot d_{\text{jk}}) \times d_h}$ is the trainable projection matrix.

$$\mathbf{H}^{(\text{JK-final})} = \mathbf{H}^{(\text{JK})} \mathbf{W}_{\text{JK}} \quad \in \mathbb{R}^{|\mathcal{V}| \times d_h} \tag{6.32}$$

The JK mechanism aggregates features across layers, preserves early-layer representations to reduce over-smoothing, and enhances features required for our multitask prediction.

6.6.4 Global Attention, Virtual Slots, and FiLM Modulation

To enhance long-range dependency modeling and enable context-aware representations in our hybrid GNN model, we incorporate three complementary mechanisms: global multi-head attention (MHA), virtual slots, and feature-wise linear modulation (FiLM). Section 6.6.4.1, Section 6.6.4.2 and Section 6.6.4.3 describe each component. Together, these components form a hybrid global-local architecture:

- **MHA** [VSP+17] captures direct pairwise global dependencies between nodes.
- **Virtual slots** [HLG+19] serve as compact recurrent memory units that summarize the graph.
- **FiLM** [PSDV+18] injects global contextual signals back into node-level computations.

This combination ensures that the network can model both fine-grained local relations and high-level global dependencies, which is crucial for tasks such as edge prediction, Steiner point identification, and global structural reasoning.

6.6.4.1 Global Multi-Head Attention

We employ a global multi-head attention (MHA) layer that operates across all node embeddings in Equation 6.33 where head_j is defined in Equation 6.34 and $\mathbf{H} \in \mathbb{R}^{|\mathcal{V}| \times d_h}$ is the matrix of node embeddings, h is the number of heads, and $\mathbf{W}_j^Q, \mathbf{W}_j^K, \mathbf{W}_j^V \in \mathbb{R}^{d_h \times d_h}$ are learnable projections. This allows each node to attend to all other nodes globally, thereby capturing long-range interactions beyond the local message passing

neighborhood.

$$\mathrm{MHA}(\mathbf{H}) = \mathbf{concat}\left(\mathrm{head}_1, \mathrm{head}_2, \ldots, \mathrm{head}_h\right)\mathbf{W}^O \tag{6.33}$$

$$\mathrm{head}_j = \mathrm{softmax}\left(\frac{\mathbf{H}\mathbf{W}_j^Q(\mathbf{H}\mathbf{W}_j^K)^\top}{\sqrt{d_h}}\right)\mathbf{H}\mathbf{W}_j^V, \tag{6.34}$$

6.6.4.2 Virtual Slots with Gated Recurrent Unit Update

Virtual slots aggregate information across nodes into a compact set of latent variables, enabling the model to store and refine global context that is shared across the graph. Hence, we introduce a fixed number of virtual slots $\mathbf{s} \in \mathbb{R}^{M \times d_h}$ that act as global memory units. Node embeddings attend to slots via query–key–value attention, and the slot states are recurrently updated using a gated recurrent unit (GRU) described in Equation 6.36 where $\mathbf{s}_s$ is the state of the s-th slot.

$$\alpha_{i,s} = \mathrm{softmax}_s\left(\frac{(\mathbf{h}_i\mathbf{W}^Q)(\mathbf{s}_s\mathbf{W}^K)^\top}{\sqrt{d_h}}\right) \tag{6.35}$$

$$\mathbf{u}_s = \sum_{i \in \mathcal{V}} \alpha_{i,s}(\mathbf{h}_i\mathbf{W}^V), \quad \mathbf{s}_s^{(t+1)} = \mathrm{GRU}(\mathbf{u}_s, \mathbf{s}_s^{(t)}). \tag{6.36}$$

6.6.4.3 Feature-wise Linear Modulation (FiLM)

FiLM enables fine-grained conditioning of node features on global context, ensuring that local node updates are modulated according to the learned global state of the graph. To inject global context back into node embeddings, we employ FiLM modulation as $\mathbf{h}_i' = \gamma(\mathbf{c}) \odot \mathbf{h}_i + \beta(\mathbf{c})$ where $\mathbf{c} \in \mathbb{R}^{d_h}$ is a context vector (e.g., derived from aggregated slot representations), and $\gamma(\mathbf{c}), \beta(\mathbf{c}) \in \mathbb{R}^{d_h}$ are learned affine transformations.

6.6.5 Adaptive Edge-Level Attention

We use adaptive edge-level attention to enhance message-passing. Each edge feature vector $\mathbf{f}_{ij}^{\mathrm{out}} \in \mathbb{R}^{d_h}$ of edge (i, j) is linearly projected into three distinct representations:

a *query*, a *key*, and a *value*. This is implemented via independent linear transformations: $\mathbf{q}_i = \mathbf{W}_q \cdot \mathbf{h}_i + \mathbf{b}_q$, $\mathbf{k}_{ij} = \mathbf{W}_k \cdot \mathbf{f}_{ij}^{\text{out}} + \mathbf{b}_k$ and $\mathbf{v}_{ij} = \mathbf{W}_v \cdot \mathbf{f}_{ij}^{\text{out}} + \mathbf{b}_v$, where $\mathbf{q}_i \in \mathbb{R}^{d_h}$ is a query vector derived from the node feature $\mathbf{h}_i$, while $\mathbf{k}_{ij}, \mathbf{v}_{ij} \in \mathbb{R}^{d_h}$ are the key and value vectors derived from the edge feature. The learnable matrices $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{d_h \times d_h}$ and biases $\mathbf{b}_q, \mathbf{b}_k, \mathbf{b}_v \in \mathbb{R}^{d_h}$ parameterize three independent linear layers. The attention score α_{ij} between node i and edge (i, j) is then computed as shown in Equation 6.37.

$$\alpha_{ij} = \text{softmax}_{j \in \texttt{NBR}(i)} \left(\frac{\mathbf{q}_i^\top \mathbf{k}_{ij}}{\sqrt{d_h}} \right), \tag{6.37}$$

Finally, the message passed from edge (i, j) to node i is weighted by α_{ij} as shown in Equation 6.38. Aggregating edge features provides both node-level summaries m_i, and mean edge descriptors $\bar{\mathbf{v}}_i = \text{mean}_{j:(i,j)\in\mathcal{E}}(\mathbf{v}_{ij})$.

$$\mathbf{m}_i = \sum_{j \in \texttt{NBR}(i)} \alpha_{ij} \, \mathbf{v}_{ij}. \tag{6.38}$$

This formulation allows the model to (i) adaptively focus on more informative edges via α_{ij} and (ii) propagate edge-specific contextualized information through $\mathbf{v}_{ij}$. The separation of query, key, and value projections parallels the design of self-attention mechanisms, while being explicitly tailored to edge-aware graph message passing.

6.6.6 Modeling Inter-Net Dependencies

Capturing inter-net dependencies is one of the primary motivations behind the proposed framework. Unlike existing methods that construct RSMTs independently for individual nets, the proposed approach forms a combined Hanan-grid graph containing multiple nets simultaneously. In this representation, nodes and edges contain net-membership indicators that explicitly encode associations with multiple nets. Inter-net dependencies are captured through several mechanisms.

First, message passing is performed on the combined Hanan-grid graph rather than on isolated net graphs. Consequently, neighboring structures belonging to different nets influence each other's learned representations during feature propagation.

Second, the ResGateGCN layers utilize edge-aware gating mechanisms, allowing information from neighboring nodes and shared routing regions to be selectively propagated through the graph.

Third, the ResGATv2 layers dynamically assign attention weights to neighboring nodes based on their contextual importance. This enables the model to identify routing regions where interactions among multiple nets are particularly significant.

Fourth, MultiHop aggregation extends information propagation beyond immediate neighbors, enabling the model to capture higher-order interactions among routing structures belonging to different nets.

Finally, global multi-head attention, virtual slots, and FiLM modulation provide graph-level contextual information. These mechanisms allow routing decisions for one net to be influenced by the global routing structure formed by all nets within the group.

6.6.7 Node Classifiers

To apply node classifiers, at first, a shared node feature is built using the final JK-projected node embeddings, i.e., $\mathbf{h}_i^{'}$, node slot context $\mathbf{u}_i$, node edge summary $\mathbf{m}_i$, and mean edge descriptors $\bar{\mathbf{v}}_i$ as defined in Equation 6.39. After that, we apply an MLP layer with shared node representation, followed by two parallel classifiers to predict the connecting node and Steiner point.

$$\zeta_i = \left[\mathbf{h}_i^{'}, \mathbf{u}_i, \mathbf{m}_i, \bar{\mathbf{v}}_i\right] \in \mathbb{R}^{|\mathcal{V}|\times(4\cdot d_h)}. \tag{6.39}$$

6.6.7.1 Shared node MLP

The shared MLP maps the concatenated node features $\mathbf{f} \in \mathbb{R}^{|\mathcal{V}|\times(4\cdot d_h)}$ into a condensed shared representation $\tilde{\mathbf{h}} \in \mathbb{R}^{|\mathcal{V}|\times d_h}$ and it is defined in Equation 6.40 where $\mathbf{W}_s \in \mathbb{R}^{d_h\times 4d_h}$ and $b_s \in \mathbb{R}^{d_h}$ are learnable parameters. Concretely, for each node i, $\tilde{\mathbf{h}}_i$ is defined in Equation 6.41.

$$\tilde{\mathbf{h}} = \mathrm{Dropout}_{0.3}\Big(\mathrm{ReLU}\big(\mathrm{LN}\big(\ell_{W_s,b_s}(\zeta)\big)\big)\Big) \tag{6.40}$$

$$\tilde{\mathbf{h}}_i = \text{Dropout}_{0.3}\Big(\text{ReLU}\big(\text{LN}\big(W_s\zeta_i + b_s\big)\big)\Big) \in \mathbb{R}^{d_h}. \tag{6.41}$$

6.6.7.2 Connecting node classifier

This head maps the shared representation $\tilde{\mathbf{h}} \in \mathbb{R}^{|\mathcal{V}|\times d_h}$ to per-node logits for the useful connecting node prediction. It is a two-layer MLP with a half-sized hidden layer and dropout as shown in Equation 6.42 where $\mathbf{W}_c^{(1)} \in \mathbb{R}^{(d_h/2)\times d_h}$, $\mathbf{b}_c^{(1)} \in \mathbb{R}^{d_h/2}$ and $\mathbf{W}_c^{(2)} \in \mathbb{R}^{K\times(d_h/2)}$, $\mathbf{b}_c^{(2)} \in \mathbb{R}^K$.

$$\begin{aligned} \theta_i &= \text{ReLU}\big(\mathbf{W}_c^{(1)}\,\tilde{\mathbf{h}}_i + \mathbf{b}_c^{(1)}\big) \in \mathbb{R}^{d_h/2}, \\ \hat{\mathbf{y}}_i^{(\text{con_logits})} &= \mathbf{W}_c^{(2)}\,\Big(\text{Dropout}_{0.3}(\theta_i)\Big) + \mathbf{b}_c^{(2)} \in \mathbb{R}^K. \end{aligned} \tag{6.42}$$

6.6.7.3 Steiner-point Classifier

It follows the same architecture as the connecting node classifier described in Equation 6.43.

$$\begin{aligned} \vartheta_i &= \text{ReLU}\big(\mathbf{W}_s^{(1)}\,\tilde{\mathbf{h}}_i + \mathbf{b}_s^{(1)}\big) \in \mathbb{R}^{d_h/2}, \\ \hat{\mathbf{y}}_i^{(\text{stp_logits})} &= \mathbf{W}_s^{(2)}\,\Big(\text{Dropout}_{0.3}(\vartheta_i)\Big) + \mathbf{b}_s^{(2)} \in \mathbb{R}^K. \end{aligned} \tag{6.43}$$

6.6.8 Edge Classification Head

The model uses a hybrid edge decoder that combines local and global node-edge context with a learned gating mechanism. This predicts edge existence or type by leveraging local pairwise structure and global graph context, with the gate dynamically balancing their contributions for improved accuracy. A learnable positional encoding projection aligns node embeddings to a fixed dimension for downstream prediction.

6.6.8.1 Local Edge Decoder

Let two nodes i and j have embeddings $\mathbf{h}_i, \mathbf{h}_j \in \mathbb{R}^{d_h}$, and edge features $\mathbf{f}_{ij} \in \mathbb{R}^{d_h}$, we form a local feature vector as shown in Equation 6.44. This is passed through a

single-layer MLP as shown in Equation 6.45.

$$\mathbf{y}_{ij}^{\text{local}} = \mathbf{concat}(\mathbf{h}_i, \mathbf{h}_j, \mathbf{f}_{ij}) \quad \in \mathbb{R}^{3d_h} \tag{6.44}$$

$$\mathbf{o}_{ij}^{\text{local}} = \text{ReLU}(\mathbf{W}_{\text{local}} \cdot \mathbf{y}_{ij}^{\text{local}} + \mathbf{b}_{\text{local}}) \quad \in \mathbb{R}^{d_h} \tag{6.45}$$

6.6.8.2 Global Edge Decoder

To embed global structure awareness, we use a graph-level context vector $\mathbf{c}_G \in \mathbb{R}^{d_h}$, typically derived via global mean pooling over all node features. We concatenate this with the local inputs as shown in Equation 6.46. This is passed through a similar MLP as shown in Equation 6.47.

$$\mathbf{y}_{ij}^{\text{global}} = \mathbf{concat}(\mathbf{h}_i, \mathbf{h}_j, \mathbf{c}_G, \mathbf{e}_{ij}) \quad \in \mathbb{R}^{4d_h} \tag{6.46}$$

$$\mathbf{o}_{ij}^{\text{global}} = \text{ReLU}(\mathbf{W}_{\text{global}} \cdot \mathbf{y}_{ij}^{\text{global}} + \mathbf{b}_{\text{global}}) \quad \in \mathbb{R}^{d_h} \tag{6.47}$$

6.6.8.3 Edge Gating Mechanism

Edge Gate is used to dynamically weight the contribution of local vs. global features based on the specific edge input, instead of statically choosing one. This improves adaptability across diverse graph topologies. We construct a gate input using the same components as the local decoder shown in Equation 6.48. This passes through a LayerNorm and Dropout-regularized MLP, ending with a sigmoid to produce a gate scalar as shown in Equation 6.49. Then we blend local and global outputs as shown in Equation 6.50.

$$\mathbf{y}_{ij}^{\text{gate}} = \mathbf{concat}(\mathbf{h}_i, \mathbf{h}_j, \mathbf{e}_{ij}) \quad \in \mathbb{R}^{3d_h} \tag{6.48}$$

$$\alpha_{ij} = \sigma\big(\mathbf{w}_{\text{gate}}^{\top} \cdot \text{Dropout}\big(\text{ReLU}(\text{LN}(\mathbf{W}_{\text{gate}} \cdot \mathbf{y}_{ij}^{\text{gate}}))\big)\big) \tag{6.49}$$

$$\mathbf{o}_{ij} = \alpha_{ij} \cdot \mathbf{o}_{ij}^{\text{local}} + (1 - \alpha_{ij}) \cdot \mathbf{o}_{ij}^{\text{global}} \tag{6.50}$$

6.6.8.4 Final Edge Representation and Prediction

The final edge representation is a convex combination as shown in Equation 6.51. The prediction layer then maps this representation to the target edge dimension K, representing edge classes or existence labels (Equation 6.52).

$$\mathbf{o}_{ij} = \alpha_{ij} \cdot \mathbf{o}_{ij}^{\text{local}} + (1 - \alpha_{ij}) \cdot \mathbf{o}_{ij}^{\text{global}} \tag{6.51}$$

$$\hat{\mathbf{y}}_{ij}^{(\text{edge_logits})} = \mathbf{W}_{\text{out}} \cdot \mathbf{o}_{ij} + \mathbf{b}_{\text{out}} \in \mathbb{R}^K \tag{6.52}$$

6.6.9 Discussion on the Selection of the GNN Architecture

The proposed framework employs a hybrid GNN architecture consisting of Residual Gated Graph Convolutional Networks (ResGateGCN), Residual Graph Attention Network v2 (ResGATv2), MultiHop aggregation layers, Jumping Knowledge aggregation, Global Multi-Head Attention, Virtual Slots, Feature-wise Linear Modulation (FiLM), and Adaptive Edge-Level Attention.

Alternative GNN architectures were considered conceptually during model design. A conventional Graph Convolutional Network (GCN) provides efficient neighborhood aggregation but suffers from limited expressiveness in capturing edge-dependent interactions and may experience over-smoothing when deeper architectures are used. In contrast, ResGateGCN incorporates edge-aware gating mechanisms and residual connections, enabling more effective modeling of routing graph structures.

Graph Attention Networks (GATs)[VCC+17] dynamically weight neighboring nodes and improve representation learning. Consequently, GATv2 [BAY21] layers were incorporated into the proposed framework to complement the structural learning capability of ResGateGCN. However, pure GAT-based architectures generally incur higher computational overhead and may not effectively model explicit edge features without additional modifications.

GraphSAGE[HYL17] provides scalable neighborhood sampling and efficient training on large graphs. However, its aggregation mechanisms are relatively simple and may not adequately capture the complex node-edge interactions required for concurrent RSMT construction.

Graph Transformer architectures [DB20] can model long-range dependencies more effectively through global attention mechanisms. However, their computational complexity increases significantly with graph size. To balance expressiveness and scalability, the proposed framework incorporates global multi-head attention and virtual slots to capture long-range dependencies without requiring a full graph-transformer architecture. Therefore, the proposed hybrid design combines the complementary strengths of gated convolutions, attention mechanisms, higher-order neighborhood aggregation, and global-context modeling while maintaining practical training and inference complexity.

6.6.10 Loss Generation

Our hybrid GNN model is jointly optimized for three primary tasks: (a) edge prediction and (b) Connecting node classification. (c) Steiner point classification. All tasks suffer from significant class imbalance—only a small fraction of edges belong to the RSMTs compared to the total edges of the Hanan grid, and likewise, connecting nodes, especially Steiner points, are far fewer than the total number of nodes. To address this imbalance and ensure the model focuses on hard-to-classify, informative samples, we employ Focal Loss [LGG+17]. The focal loss is defined in Equation 6.53.

$$\mathcal{FL}(\hat{\mathbf{y}}^{(\mathbf{logits})}, \mathbf{y}; \delta, \gamma) = \delta(1-p_t)^{\gamma}\mathcal{L}_{\text{BCE}}(\hat{\mathbf{y}}^{(\mathbf{logits})}, \mathbf{y}) \tag{6.53}$$

where, $\hat{\mathbf{y}}^{(\mathbf{logits})} \in \mathbb{R}^n$ is the predicted logits, $\mathbf{y} \in \{0,1\}^n$ is the binary ground-truth labels, $\mathcal{L}_{\text{BCE}}$ is the binary cross-entropy with logits and $p_t = e^{-\mathcal{L}_{\text{BCE}}}$ represents the true class predicted probability. $\gamma \geq 0$ is the focusing parameter, and $\delta \in [0,1]$ is a class-balance factor dynamically computed as $\delta = \frac{\mathbf{w_{pos}}}{\mathbf{w_{pos}}+1}$ where $\mathbf{w_{pos}} = \frac{\sum(y=0)}{\sum(y=1)+\varepsilon}$. The edge, useful connecting node, and Steiner point prediction losses are shown in Equation 6.54, Equation 6.55 and Equation 6.56 respectively, where, $\mathbf{y}^{\text{edge}} \in \{0,1\}^{|\mathcal{E}|\times K}$, $\mathbf{y}^{\text{con}} \in \{0,1\}^{|\mathcal{V}|\times K}$, and $\mathbf{y}^{\text{stp}} \in \{0,1\}^{|\mathcal{V}|\times K}$ are actual probabilities and δ_e, δ_{con} and δ_{stp} are the class-balance factor for edge, connecting node, and Steiner point, respectively.

$$\mathcal{L}_{\text{edge}} = \frac{1}{K|\mathcal{E}|}\sum_{i=1}^{K}\sum_{j=1}^{|\mathcal{E}|}\mathcal{FL}(\hat{\mathbf{y}}_{ij}^{(\text{edge_logits})}, \mathbf{y}_{ij}^{(edge)}; \delta_e, \gamma) \tag{6.54}$$

$$\mathcal{L}_{\text{con}} = \frac{1}{K|\mathcal{V}|} \sum_{i=1}^{K} \sum_{j=1}^{|\mathcal{V}|} \mathcal{FL}(\hat{\mathbf{y}}_i^{(\text{con_logits})}, \mathbf{y}_i^{(con)}, \delta_{con}, \gamma) \tag{6.55}$$

$$\mathcal{L}_{\text{stp}} = \frac{1}{K|\mathcal{V}|} \sum_{i=1}^{K} \sum_{j=1}^{|\mathcal{V}|} \mathcal{FL}(\hat{\mathbf{y}}_i^{(\text{stp_logits})}, y_i^{(stp)}, \delta_{stp}, \gamma) \tag{6.56}$$

We also applied several regularization losses. The wirelength loss (Equation 6.57) encourages the predicted RSMT to match the ground-truth total length, where ℓ_{ij} denotes edge length. The context loss $\mathcal{L}_{\text{ctx}}$ aligns latent node-slot representations with JK-projected features, while positional encoding (PE) loss preserves relative distances by minimizing the MSE between projected and raw PEs. The final training loss is a weighted sum of all task-specific and regularization losses (Equation 6.58).

$$\mathcal{L}_{\text{wl}} = \Big| \sum_{(i,j)\in\mathcal{E}} \hat{\mathbf{y}}_{ij}^{\text{edge_logits}} \cdot \ell_{ij} - \sum_{(i,j)\in\mathcal{E}} \mathbf{y}_{ij}^{\text{edge}} \cdot \ell_{ij} \Big| \tag{6.57}$$

$$\mathcal{L}_{\text{total}} = \lambda_{\text{edge}} \cdot \mathcal{L}_{\text{edge}} + \lambda_{\text{con}} \cdot \mathcal{L}_{\text{con}} + \lambda_{\text{stp}} \cdot \mathcal{L}_{\text{stp}} + \lambda_{\text{wl}} \cdot \mathcal{L}_{\text{wl}} + \lambda_{\text{ctx}} \cdot \mathcal{L}_{\text{ctx}} + \lambda_{\text{pe}} \cdot \mathcal{L}_{\text{pe}}. \tag{6.58}$$

6.7 Formation of Individual Predicted RSMTs

During testing, our hybrid GNN model predicts edge logits $\hat{\mathbf{Y}}^{(\text{edge_logits})} \in \mathbb{R}^{|\mathcal{E}|\times K}$, useful connecting node logits $\hat{\mathbf{Y}}^{(\text{con_logits})} \in \mathbb{R}^{|\mathcal{V}|\times K}$, and Steiner point logits $\hat{\mathbf{Y}}^{(\text{stp_logits})} \in \mathbb{R}^{|\mathcal{V}|\times K}$. Applying the logistic sigmoid activation, we obtain probabilistic scores:

$$\hat{\mathbf{Y}}^{(\text{edge})} = \sigma\left(\hat{\mathbf{Y}}^{(\text{edge_logits})}\right) \tag{6.59}$$

$$\hat{\mathbf{Y}}^{(\text{con})} = \sigma\left(\hat{\mathbf{Y}}^{(\text{con_logits})}\right) \tag{6.60}$$

$$\hat{\mathbf{Y}}^{(\text{stp})} = \sigma\left(\hat{\mathbf{Y}}^{(\text{stp_logits})}\right) \tag{6.61}$$

6.7.1 Branch-Aware Edge Probability Smoothing

For each edge $(i, j) \in \mathcal{E}$, define its geometric orientation as either *horizontal* or *vertical*. Edges sharing the same orientation and aligned coordinate (e.g., same x_i for vertical, or same y_i for horizontal) form a *branch group* $\mathcal{B}$. For each branch $\mathcal{B}$, the raw edge probabilities $\{\hat{\mathbf{y}}_{ij}^{(edge)} \mid (i, j) \in \mathcal{B}\}$ are averaged to yield a smoothed branch probability defined in Equation 6.62. This smoothing reduces local noise and keeps straight segments geometrically consistent.

$$\text{avg}(\hat{\mathbf{y}}_{ij}^{(edge)}) = \frac{1}{|\mathcal{B}|} \sum_{(i,j)\in\mathcal{B}} \hat{\mathbf{y}}_{ij}^{(edge)} \quad \forall (i, j) \in \mathcal{B} \tag{6.62}$$

6.7.2 Priority-Guided Tree Construction

For each net N_k, we define an edge priority score that balances geometric cost and probabilities as defined in Equation 6.63 where d_{ij} is the edge length of edge $(i, j) \in \mathcal{E}$.

$$\pi_{ij}^{(N_k)} = \frac{d_{ij}}{\text{avg}(\hat{\mathbf{y}}_{ij}^{(edge)}) + \mathbf{con_w} + \mathbf{stp_w} + \epsilon} \tag{6.63}$$

where $\mathbf{con_w} = \lambda_1(\hat{\mathbf{y}}_i^{(con)} + \hat{\mathbf{y}}_j^{(con)})$, $\mathbf{stp_w} = \lambda_2(\hat{\mathbf{y}}_i^{(stp)} + \hat{\mathbf{y}}_j^{(stp)})$, λ_1 and λ_2 are weighting factors controlling the influence of connecting-node probabilities and Steiner point probabilities, and ϵ is a small constant for numerical stability.

Edges are sorted in ascending order of $\pi_{ij}^{(N_k)}$, forming a candidate list $\mathcal{L}^{(N_k)}$. Using a union–find (disjoint-set) data structure, a minimal subset of edges $\hat{\mathcal{E}}^{(N_k)} \subseteq \mathcal{E}$ is selected such that all terminals of the net N_k become connected. This process yields a predicted RSMT $\hat{\mathcal{G}}^{(N_k)} = (\mathcal{V}, \hat{\mathcal{E}}^{(N_k)})$.

6.8 Experimental Setup

In our approach, we generate RSMTs for 4 nets concurrently, i.e., $K = 4$. Consequently, each combined Hanan grid graph and its corresponding edge-overlap-minimized combined RSMT represent 4 nets. In the combined Hanan grid graph, the dimen-

sions of node and edge feature vectors are $d_v = 2 + K = 6$, and $d_e = 1 + K = 5$, respectively. In the ground truth combined RSMT, the dimensions of node and edge labels are $d_v = 2 + 3K = 14$, and $d_e = 1 + K = 5$, respectively.

In our model, we consider the hidden dimension $d_h = 96$ and message passing layers $L = 10$, comprising 4 residual gated GCN layers, 4 residual GATv2 convolutional layers, and 2 multi-hop aggregation layers as shown in Figure 6.6. Details of the training and validation datasets are provided in Section 6.8.2, while the testing dataset is described in Section 6.8.3.

6.8.1 Selection of Hyperparameters

The proposed framework contains several important hyperparameters including hidden dimension ($d_h = 96$), number of message-passing layers ($L = 10$), MultiHop interval ($I = 4$), focal-loss weighting parameters, optimizer learning rate, and loss balancing coefficients. These hyperparameters were selected empirically through repeated validation experiments using the validation dataset.

Several architectural choices were motivated by stability considerations. Residual connections, layer normalization, Jumping Knowledge aggregation, and FiLM modulation were incorporated specifically to reduce sensitivity to network depth and improve training robustness.

Experimental observations indicate that moderate variations in learning rate, hidden dimension, and loss weights do not significantly alter the final routing quality, suggesting reasonable robustness of the proposed architecture.

Furthermore, the consistent performance across ten independently generated test datasets indicates that the selected hyperparameter configuration generalizes well and is not excessively sensitive to minor parameter variations.

6.8.2 Details of Training Dataset

Our training dataset consists of pairs of combined Hanan grid graphs and their corresponding combined RSMT graphs. We train our model using nets with degrees ranging from 10 to 50. Nets with degrees below 10 are excluded, as FLUTE [CW07]

can efficiently generate their optimal RSMTs using a precomputed lookup table. Due to computational resource constraints, we limit the maximum net degree to 50. Since both the combined Hanan grid graphs and the combined RSMT graphs are constructed from multiple nets, their dimensions may scale with the degrees of the constituent nets.

As discussed in Section 6.4, the training and validation datasets are generated by first randomly creating nets with degrees ranging from 10 to 50. These nets are then grouped using the Net Grouping procedure, using the method discussed in Section 6.5.1. To obtain the optimal RSMTs for each individual net within a group, we use `GeoSteiner` [JWWZ18]. Subsequently, we construct group-wise combined Hanan grid graphs and the corresponding combined RSMT graphs and use ILP to mitigate edge overlap and form the final training and validation datasets using the methods discussed in Section 6.5.2 and Section 6.5.3, respectively.

For training, we generate a total of 10000 nets with degrees ranging from 10 to 50. To ensure a balanced dataset containing high-, medium-, and low-degree nets, we divide the nets into four pools, each covering a specific degree range and contributing a fixed proportion of the total dataset.

- `Pool 1:` Nets with degrees 10–20, contributing 40% of the total nets.
- `Pool 2:` Nets with degrees 21–30, contributing 30%.
- `Pool 3:` Nets with degrees 31–40, contributing 20%.
- `Pool 4:` Nets with degrees 41–50, contributing 10%.

This distribution reflects empirical observations that high-degree nets occur less frequently than medium-degree nets in practical scenarios. Although we experimented with alternative distributions, this configuration yielded the most stable and effective results. While the number of nets selected from each `pool` was fixed, the specific nets within each pool were generated randomly. For concurrent RSMT construction, we group nets into sets of $k = 4$, resulting in $\frac{10000}{4} = 2500$ combined samples. Each sample contains a combined Hanan grid graph and its corresponding edge-overlap-minimized combined RSMT graph.

Similarly, for validation, we generate 2000 nets with degrees ranging from 10 to 50, following the same degree distribution as in the training set. Using the same group size of $k = 4$, we obtain $\frac{2000}{4} = 500$ validation samples, each comprising a combined

Table 6.6: Test Dataset

Test Dataset	Total #Nets	% #Nets by degree			
		10–20	21–30	31–40	41–50
TestDS-1	5000	40	30	20	10
TestDS-2	5000	40	30	20	10
TestDS-3	5000	40	30	20	10
TestDS-4	5000	40	30	20	10
TestDS-5	5000	50	20	20	10
TestDS-6	5000	40	40	10	10
TestDS-7	5000	30	50	10	10
TestDS-8	5000	20	40	30	10
TestDS-9	5000	30	30	30	10
TestDS-10	5000	30	30	20	20

Hanan grid graph and its corresponding edge-overlap-minimized combined RSMT.

6.8.3 Details of Test Dataset

Our test dataset consists solely of combined Hanan grid graphs, without access to the corresponding RSMTs. For testing, we consider 10 datasets, each containing nets with varying degrees as summarized in Table 6.6. The first four test datasets share the same degree distribution across pools. However, since the nets within each pool are generated randomly, the specific nets differ across datasets even when their degree proportions remain identical. For each test dataset, we use the same group size of $k = 4$, forming $\frac{5000}{4} = 1250$ groups. A combined Hanan grid graph is constructed for each group.

6.9 Experimental Results

We have implemented our hybrid GNN model for concurrent construction of RSMTs of multiple nets using `PyTorch 2.6`. We train and test our model on a 64-bit Linux

machine with an NVIDIA A100 GPU with 40 GB memory. As explained in Section 6.8, we train our model with 2500 pairs of a combined Hanan grid graph and its corresponding edge-overlap-minimized combined RSMT graph.

We train our model for 100 epochs each with batch size 1. We use the Adam optimizer with a learning rate of 5×10^{-4} and weight decay of 1×10^{-4}. We consider $\lambda_{edge} = 2.0$, $\lambda_{con} = 2.0$, $\lambda_{stp} = 10.0$, $\lambda_{wl} = 0.001$, $\lambda_{ctx} = 1.0$ and $\lambda_{pe} = 1.0$ in Equation 6.58 as weights. We use a learning rate scheduling strategy that reduces the learning rate by a factor of 0.75 if the validation loss does not improve for 3 consecutive epochs. This helps the model converge more effectively by enabling larger updates early on and finer adjustments as training plateaus.

After training, we evaluate our model using the test datasets described in Table 6.6. For each test dataset, we generate edge and node prediction vectors for all nodes and edges in the group-wise combined Hanan grid graphs. Existing popular RSMT construction techniques—such as `GeoSteiner` (optimal) [JWWZ18], `Flute` [CW07], `REST` [LCY21], and `cGAN-RSMT` [SBSK25]—build RSMTs in a sequential (one-by-one) manner. As a result, they are unable to explicitly minimize edge overlaps during RSMT construction.

In contrast, our proposed approach constructs multiple RSMTs concurrently, inherently reducing edge overlap during the construction process itself. However, since existing methods do not include such overlap minimization, a direct comparison would be unfair.

To establish a fair comparison, we therefore apply our ILP-based edge-overlap minimization (`IEOM`) procedure described in Section 6.5.3.1 to the RSMTs generated by the existing methods (`GeoSteiner` (optimal) [JWWZ18], `Flute` [CW07], `REST` [LCY21], and `cGAN-RSMT` [SBSK25]). Therefore, we form the following baseline methods that serve as the baseline methods for evaluating the effectiveness of our proposed concurrent, overlap-aware RSMT construction method.

- Baseline 1: `GeoSteiner`[JWWZ18] and `IEOM`
- Baseline 2: `Flute`[CW07] and `IEOM`
- Baseline 3: `REST`[LCY21] and `IEOM`
- Baseline 4: `cGAN-RSMT` [SBSK25] and `IEOM`

Table 6.7 shows the comparison of our method with the baselines on average

Table 6.7: Comparison of avg. % error in wirelength of our method with Baseline 1: `GeoSteiner`[JWWZ18] + `IEOM`*, Baseline 2: `Flute`[CW07] + `IEOM`*, Baseline 3: `REST`[LCY21] + `IEOM`* and Baseline 4: `cGAN-RSMT` [SBSK25] + `IEOM`*

Test Dataset	Baseline 1: `GeoSteiner`[JWWZ18] +`IEOM`*	Baseline 2: `Flute`[CW07] +`IEOM`*	Baseline 3: `REST`[LCY21] +`IEOM`*	Baseline 4: `cGAN-RSMT`[SBSK25] +`IEOM`*	Ours
`TestDS-1`	0.0	0.1994	0.1678	0.1618	0.2042
`TestDS-2`	0.0	0.221	0.1145	0.1213	0.1793
`TestDS-3`	0.0	0.2038	0.1643	0.1612	0.1945
`TestDS-4`	0.0	0.202	0.151	0.1542	0.1976
`TestDS-5`	0.0	0.2319	0.2013	0.1818	0.1934
`TestDS-6`	0.0	0.2434	0.2341	0.232	0.2351
`TestDS-7`	0.0	0.2832	0.264	0.251	0.2683
`TestDS-8`	0.0	0.3402	0.307	0.316	0.3321
`TestDS-9`	0.0	0.432	0.3613	0.3235	0.3472
`TestDS-10`	0.0	0.4731	0.4191	0.402	0.4474
`avg.`	0.0	0.283	0.2384	0.2304	0.2599

*`IEOM: ILP-based Edge Overlap Mitigation`

Table 6.8: Comparison of RSMT Generation Time (sec) of our method with Baseline 1: `GeoSteiner` [JWWZ18] + `IEOM`*, Baseline 2: `Flute` [CW07] + `IEOM`*, Baseline 3: `REST` [LCY21] + `IEOM`* and Baseline 4: `cGAN-RSMT` [SBSK25] + `IEOM`*

Test Dataset	Baseline 1: `GeoSteiner`[JWWZ18] +`IEOM`*	Baseline 2: `Flute`[CW07] +`IEOM`*	Baseline 3: `REST`[LCY21] +`IEOM`*	Baseline 4: `cGAN-RSMT`[SBSK25] +`IEOM`*	Ours
`TestDS-1`	$25.34 + 128.89 = 154.23$	$16.92 + 130.03 = 146.95$	$11.92 + 129.53 = 141.45$	$13.12 + 129.46 = 142.58$	7.34
`TestDS-2`	$26.02 + 129.4 = 155.42$	$17.04 + 128.9 = 145.94$	$11.04 + 129.2 = 140.24$	$13.92 + 129.54 = 143.46$	7.73
`TestDS-3`	$25.54 + 129.38 = 154.92$	$17.03 + 129.08 = 146.11$	$11.33 + 128.18 = 139.51$	$12.92 + 128.31 = 141.23$	7.65
`TestDS-4`	$25.67 + 129.76 = 155.43$	$17.67 + 128.96 = 146.63$	$12.07 + 128.76 = 140.83$	$13.02 + 129.7 = 142.72$	7.81
`TestDS-5`	$24.1 + 128.23 = 152.33$	$16.1 + 127.76 = 143.86$	$10.23 + 128.36 = 138.59$	$12.92 + 128.03 = 140.95$	6.92
`TestDS-6`	$27.98 + 129.56 = 157.54$	$18.34 + 128.93 = 147.27$	$13.45 + 129.13 = 142.58$	$13.92 + 128.8 = 142.72$	8.03
`TestDS-7`	$29.34 + 130.52 = 159.86$	$18.16 + 133.52 = 151.68$	$12.53 + 131.22 = 143.75$	$13.05 + 132.13 = 145.18$	8.67
`TestDS-8`	$35.74 + 132.3 = 168.04$	$20.24 + 131.3 = 151.54$	$13.14 + 130.53 = 143.67$	$15.87 + 130.45 = 146.32$	9.02
`TestDS-9`	$43.32 + 158.4 = 201.72$	$26.71 + 157.4 = 184.11$	$16.7 + 157.02 = 173.72$	$20.02 + 158.02 = 178.04$	9.45
`TestDS-10`	$43.56 + 162.35 = 205.91$	$25.16 + 166.35 = 191.51$	$16.02 + 158.15 = 174.17$	$20.3 + 159.67 = 179.97$	9.35
`avg.`	166.54	155.56	147.851	150.317	8.197

*`IEOM: ILP-based Edge Overlap Mitigation`

Table 6.9: Comparison of group-wise avg. % overlap mitigation of our method with Baseline 1: `GeoSteiner` [JWWZ18] + `IEOM`*, Baseline 2: `Flute` [CW07] + `IEOM`*, Baseline 3: `REST` [LCY21] + `IEOM`* and Baseline 4: `cGAN-RSMT` [SBSK25] + `IEOM`*

Test Dataset	Baseline 1: `GeoSteiner` [JWWZ18] + `IEOM`*	Baseline 2: `Flute`[CW07] + `IEOM`*	Baseline 3: `REST`[LCY21] +`IEOM`*	Baseline 4: `cGAN-RSMT` [SBSK25] + `IEOM`*	Ours
`TestDS-1`	98.65	98.4	98.65	98.54	97.45
`TestDS-2`	100.00	99.45	100.00	99.85	98.87
`TestDS-3`	99.54	99.32	99.54	99.44	99.23
`TestDS-4`	99.43	99.49	99.45	99.54	99.05
`TestDS-5`	98.65	99.23	99.45	99.75	99.25
`TestDS-6`	99.45	99.4	99.3	99.2	98.54
`TestDS-7`	99.74	99.23	99.35	99.2	98.76
`TestDS-8`	99.43	99.13	99.05	99.35	98.85
`TestDS-9`	97.95	98.34	98.01	97.84	97.05
`TestDS-10`	98.74	98.67	98.48	98.84	97.1
`avg.`	99.16	99.066	99.128	98.155	98.415

*`IEOM: ILP-based Edge Overlap Mitigation`

%error in wirelength . As it shows, our method generates RSMTs with 0.2599% wirelength error as compared with Baseline 1 (optimal). Our method has 0.0231% less wirelength error than Baseline 2. However, 0.0215% and 0.0294% more wirelength error than Baseline 3 and Baseline 4, respectively.

Table 6.8 shows that our proposed method performs concurrent construction of $K = 4$ nets, and our method did not require any additional edge overlap mitigation technique. Therefore, our method is approximately 20.32×, 18.98×, 18.04×, and 18.33× faster than Baseline 1 to Baseline 4, respectively. Now, if we ignore the running time of `IEOM` and only consider the running time of `GeoSteiner` [JWWZ18], `Flute` [CW07], `REST` [LCY21] and `cGAN-RSMT` [SBSK25] then also due to concurrent construction strategy, our method is approximately 3.74×, 2.36×, 1.56× and 1.81× faster than `GeoSteiner` [JWWZ18], `Flute` [CW07], `REST` [LCY21] and `cGAN-RSMT` [SBSK25]. This shows that even if we are able to design a more efficient edge overlap mitigation technique than our ILP-based Edge overlap mitigation (`IEOM`), our method will still take less time than the baseline methods.

Table 6.9 shows that our method can mitigate group-wise average % edge-overlap by 98.415%. Although our method generates 0.743%, 0.651%, 0.713%, and 0.74% less edge-overlap mitigation compared to Baseline 1 to 4, respectively. However, our method generates this overlap mitigation during the RSMT construction itself without applying any additional mitigation technique like other baseline methods.

6.9.1 Generalization Capability of the Proposed Framework

To improve generalization, the training dataset was generated using randomly created nets spanning multiple degree ranges from 10 to 50. Nets were grouped using a topology-aware clustering procedure that considered net degree, spatial pin distribution, centroid-based statistics, and PCA-derived topological descriptors.

The resulting training samples therefore contain substantial variation in pin locations, Hanan-grid structures, routing topologies, and inter-net interactions.

The model was evaluated on ten independently generated test datasets containing previously unseen net configurations and varying degree distributions. Consistent performance was observed across all test datasets with low average wirelength error

and substantial edge-overlap mitigation.

Nevertheless, the present evaluation focuses primarily on synthetic routing instances and benchmark-derived net distributions. Evaluation on industrial netlists with fundamentally different placement characteristics, congestion patterns, and routing styles would provide additional insight into out-of-distribution generalization and constitutes an important future validation direction.

6.9.2 Interpretability of Learned Representations

Graph neural networks learn high-dimensional latent representations that are generally more difficult to interpret than traditional rule-based routing algorithms. Nevertheless, several architectural components of the proposed framework provide insight into the learned decision-making process.

The attention coefficients generated by ResGATv2 layers indicate the relative importance of neighboring nodes during message aggregation. Similarly, adaptive edge-level attention highlights routing edges that contribute more significantly to final predictions.

The virtual-slot mechanism aggregates graph-level information into compact memory representations, enabling the model to capture global routing context. FiLM modulation then injects this global context into node-level computations, providing an interpretable pathway through which global routing information influences local decisions.

Furthermore, the combined Hanan-grid representation explicitly preserves net identities, node roles, edge memberships, and routing connectivity. The strong correlation between predicted structures and ground-truth RSMTs, together with the substantial reduction in wirelength error and edge overlap, indicates that the learned representations capture meaningful routing relationships.

Additional interpretability studies involving attention-map visualization, feature attribution analysis, latent-space exploration, and explainable graph learning techniques could provide deeper understanding of the learned routing representations and constitute an interesting direction for future investigation.

6.10 Concluding Remarks

In this chapter, we describe a method for the concurrent construction of RSMTs for multiple nets using a hybrid GNN architecture. To enable simultaneous processing of multiple nets, we introduce two novel techniques for generating combined Hanan grid graphs as inputs, and we propose an ILP-based edge overlap mitigation technique to generate group-wise edge-overlap-minimized combined RSMT graphs as ground truth, each incorporating multiple nets. Furthermore, we propose innovative node and edge embedding schemes that integrate both handcrafted and learnable features to capture net-specific structural information. A custom loss function is formulated to effectively guide the training of our hybrid GNN model. After training, the model concurrently constructs group-wise edge-overlap minimized RSMTs for $K = 4$ nets, achieving a significant reduction in total generation time and an acceptable percentage wirelength error, and group-wise percentage edge-overlap mitigation compared to the optimal and other baseline methods for test datasets containing nets up to degree 50. Future work may explore extending this approach to handle higher-degree nets (i.e., > 50) and to enable concurrent construction of minimum-bend RSMTs across multiple nets.

CHAPTER

SEVEN

CONCLUSIONS AND FUTURE DIRECTIONS

Contents

7.1 Summary 220

7.2 Future Directions 222

The key contribution of this thesis is the advancement made in the routing phase of the VLSI physical design flow. The routing quality in MEBL is enhanced by minimizing the stitch-line-induced pattern distortion in both the stitch-avoiding global and detailed routing. Reducing stitch-line violations improves the quality of the IC fabrication and circuit performance.

The thesis also propose efficient methods for constructing Rectilinear Steiner Minimum Trees (RSMTs) — either for individual nets, minimizing wirelength error and construction time, or for concurrent construction of RSMTs for multiple nets that reduce construction time and minimizes edge overlap and wirelength error. By minimizing wirelength error and edge overlap, this thesis collectively improves overall chip efficiency.

The novel contributions of this thesis and future research directions are presented in Section 7.1 and Section 7.2, respectively, which summarize the key findings, their implications, and potential directions for further development in the future.

7.1 Summary

The following presents the chapter-wise contribution of this thesis:

- In Chapter 3, we present our proposed stitch-avoiding global routing for MEBL that effectively minimizes stitch-line-induced violations. The method integrates net-shifting, stitch-constrained ILP, and stitch-aware maze routing to reduce stitch-line violations during the global routing phase. Unlike conventional approaches that address stitch-lines only in detailed routing, our framework incorporates stitch constraints early, substantially lowering violation counts. Experimental results show that our stitch-avoiding global router, followed by a traditional detailed router *RegularRoute* [ZC12], reduces via violation and short polygon violation by an average of 17.3% and 6.21%, respectively, compared to Baseline (BoxRouter [CP06] and RegularRoute [ZC12]) with no vertical routing violation. Furthermore, our approach results in a 1.89% reduction in total routed wirelength and a 0.6% improvement in routability. Therefore, due to our stitch-avoiding global router, even conventional detailed routers produce fewer stitch-line violations when preceded by our global routing framework.
- In Chapter 4, we propose a stitch-avoiding detailed routing for MEBL that effectively mitigates stitch-line violations. The method classifies nets by routing characteristics and applies a multi-stage routing strategy to minimize violations. Experimental results show that both *Routing Framework I* (BoxRouter [CP06] + our proposed stitch-avoiding detailed router [SBSK22]) and *Routing Framework II* (our stitch-avoiding global router [SPBSK22] + our proposed stitch-avoiding detailed router [SBSK22]) significantly reduce stitch-line violations compared to the *Baseline Routing Framework* (BoxRouter [CP06]+ RegularRoute [ZC12]). Notably, *Routing Framework II* achieves the best results, forming our proposed complete stitch-avoiding routing framework for MEBL. Experimental results show that Routing Framework I achieves reductions of 16.18% in via violations and 57.71% in short polygon violations, while Routing Framework II achieves reductions of 50.54% and 43.96% in via violations and short polygon violations, respectively, when compared to the *Baseline Routing Framework* with no vertical routing violation.

- In Chapter 5, we present a supervised machine learning approach for RSMT construction as image-to-image translation with a cGAN-based model. As far as we can determine from prior studies, this marks the first verified approach for generating RSMTs via cGAN-enabled image translation. We propose two novel image-encoding schemes: one encodes the Hanan grid and pin information as the input image, while the other encodes the corresponding RSMT as the ground truth. The proposed model integrates a U-Net-based generator with a PatchGAN discriminator and employs a custom loss function for effective training. Our experiments indicate that the model produces RSMTs with an average wirelength overhead of no more than 0.348% relative to the optimal solution.
- In Chapter 6, we present a hybrid GNN approach for the concurrent construction of RSMTs for multiple nets. As far as we can determine, this represents the first technique that constructs RSMTs for several nets at once using graph neural networks. For dataset preparation, we propose methods for: (1) constructing a combined Hanan grid graph as input, and (2) generating edge-overlap-minimized combined RSMT graphs as ground truth, with edge overlaps reduced via a dedicated ILP-based method (`IEOM`). We introduce novel node and edge encoding schemes that incorporate both handcrafted and learnable features. The hybrid GNN can generate edge-overlap-minimized RSMTs for multiple nets concurrently, eliminating the need for the exponential ILP-based method. Experiments demonstrate that our proposed method performs concurrent construction of $K = 4$ nets and generates RSMTs with 0.2599% wirelength error as compared with Baseline 1 (`GeoSteiner` [JWWZ18] and `IEOM`). Our method has 0.0231% less wirelength error than Baseline 2 (`Flute` [CW07] and `IEOM`). However, our method has 0.0215% and 0.0294% more wirelength error than Baseline 3 (`REST` [LCY21] and `IEOM`) and Baseline 4 (`cGAN-RSMT` [SBSK25] and `IEOM`), respectively. Our method is approximately 20.32×, 18.98×, 18.04×, and 18.33× faster than Baselines 1 to 4, respectively. Our method can mitigate group-wise average % edge-overlap by 98.415%. It generates 0.743%, 0.651%, 0.713%, and 0.74% less edge-overlap compared to Baseline 1, Baseline 2, Baseline 3, and Baseline 4, respectively.

7.2 Future Directions

Multiple potential research directions emerge from the primary contributions of this thesis. Although our stitch-avoiding routing framework for MEBL significantly reduces stitch-line violations, certain via violations can arise due to fixed pins that cannot be eliminated in the routing phase. To minimize these, the placement stage must be stitch-avoiding and routing-aware, pointing to a natural extension in routing-aware stitch-avoiding placement. Furthermore, following stitch-avoiding detailed routing, cross-layer optimization presents another important research direction.

Additionally, our proposed stitch-avoiding global and detailed routing for MEBL can be evaluated with alternative routing frameworks or heuristics. This will enable broader insights into stitch-line mitigation across diverse circuit designs with alternate routing frameworks. Together, these efforts establish a foundation for more efficient and reliable MEBL routing.

Improving scalability is a key future direction for our cGAN-based single-net RSMT construction method. Current experiments are limited to images of dimension 128×128 and can handle nets up to degree 50 due to computational constraints. With increased computational resources, larger image dimensions would allow generating RSMTs for nets with higher degrees and larger Hanan grids. However, to handle larger images, for example 256×256 or beyond, cGAN models like `pix2pixHD` [WLZ+18] can be incorporated. An updated image encoding, as well as generator and discriminator models may be required accordingly. This represents a promising direction for future research.

For the hybrid GNN approach for multi-net RSMT construction, future work includes generating bend-optimized RSMTs. This is important as bends may require additional vias, which increases the complexity of the circuit. Apart from this, another important direction is to improve the scalability of the framework by increasing the number of nets processed concurrently.

Beyond these methodology-specific extensions, several broader research directions can further strengthen the scalability, robustness, theoretical foundations, and practical applicability of the proposed methodologies.

Although the proposed methodologies demonstrated consistent performance across

diverse benchmark circuits and routing instances, further investigation of their robustness under challenging design scenarios remains an important direction for future research. Such scenarios include highly congested routing regions, dense stitch-line distributions, very high-degree nets, and design instances that differ significantly from the training distributions of the learning-based frameworks. A comprehensive analysis of these edge cases would provide deeper insights into the generalization capability, limitations, and reliability of the proposed approaches.

The proposed learning-based frameworks incorporate architectural components that provide insights into their operation, including topology-preserving image representations in the cGAN framework and message-passing mechanisms in the GNN framework. Future work will focus on enhancing model interpretability through feature attribution, attention visualization, latent representation analysis, and uncertainty estimation, thereby improving the transparency and trustworthiness of the proposed methodologies.

The proposed methodologies employ parameters and optimization settings selected through extensive validation experiments, with empirical results indicating stable performance under moderate parameter variations. Nevertheless, a more comprehensive sensitivity analysis across a broader range of algorithmic and learning-based parameters would provide deeper insights into the influence of these parameters on routing quality, computational efficiency, and model robustness, while facilitating their adaptation to diverse VLSI design scenarios.

The proposed frameworks address computationally challenging NP-hard routing and Steiner-tree optimization problems with an emphasis on developing scalable and practically effective solutions. The thesis already presents important theoretical analyses, including computational complexity analyses of the proposed algorithms, graph-theoretic formulations, and ILP-based optimization models. Future work will focus on establishing tighter complexity bounds, formal approximation guarantees, and theoretical performance analyses for the proposed learning-based frameworks, thereby providing a stronger mathematical characterization of their empirical effectiveness.

While the proposed frameworks demonstrate strong performance on large benchmark circuits and diverse routing instances, extending their evaluation to billion-gate

designs remains an important research direction. Their scalability-oriented characteristics, including hierarchical decomposition, localized optimization, efficient inference mechanisms, and graph-based learning models, naturally support partitioning and distributed computing. Future studies will investigate the performance and scalability of the proposed methodologies on industrial-scale billion-gate designs.

Validation in industrial design environments also represents an important direction for future research. Owing to the proprietary nature of commercial EDA tools, advanced technology-node datasets, and industrial design flows, such evaluation was beyond the scope of the present work. Future work will focus on collaboration with semiconductor industries and EDA tool developers to evaluate the proposed methodologies on commercial-scale designs, assess their integration into existing physical design workflows, and facilitate their adoption in practical VLSI design and manufacturing.

The thesis already provides detailed descriptions of the proposed algorithms, benchmark datasets, model architectures, training procedures, and experimental setup and methodologies to facilitate reproducibility. Future efforts will focus on the public release of source code, trained models, benchmark generation scripts, and comprehensive hyperparameter configurations to further enhance reproducibility, encourage independent validation, and promote broader adoption and extension of the proposed methodologies by the VLSI physical design research community.

Pursuing these research directions will advance practical implementations and also support the broader adoption of a stitch-avoiding physical design flow for MEBL and machine-learning-based routing approaches.

BIBLIOGRAPHY

[ACR+04] S. Adya, S. Chaturvedi, J. Roy, D. Papa, I. Markov. Unification of partitioning, floorplanning and placement. In *Proc. ICCAD*, pages 550–557, 2004.

[AK94] C. J. Alpert, A. B. Kahng. Multi-way partitioning via spacefilling curves and dynamic programming. In *Proceedings of the 31st annual Design Automation Conference*, pages 652–657, 1994.

[ALL+20] M. B. Alawieh, W. Li, Y. Lin, L. Singhal, M. A. Iyer, D. Z. Pan. High-definition routing congestion prediction for large-scale fpgas. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 26–31. IEEE, 2020.

[AM04] S. N. Adya, I. L. Markov. Fixed-outline floorplanning: Enabling hierarchical design. *IEEE transactions on very large scale integration (VLSI) systems*, 11(6):1120–1135, 2004.

[ATS+21] R. Ahmed, M. A. Turja, F. D. Sahneh, M. Ghosh, K. Hamm, S. Kobourov. Computing steiner trees using graph neural networks. *arXiv preprint arXiv:2108.08368*, 2021.

[BAY21] S. Brody, U. Alon, E. Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

[BKK+22] I. Bustany, A. B. Kahng, I. Koutis, B. Pramanik, Z. Wang. Specpart: A supervised spectral framework for hypergraph partitioning solution improvement. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2022.

[BL17] X. Bresson, T. Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

[BPL+21] D. Beaini, S. Passaro, V. Létourneau, W. Hamilton, G. Corso, P. Liò. Directional graph networks. In *International Conference on Machine Learning*, pages 748–758. PMLR, 2021.

[CC09a] H.-Y. Chen, Y.-W. Chang. Global and detailed routing. In *Electronic Design Automation*, pages 687–749. Elsevier, 2009.

[CC09b] H.-Y. Chen, Y.-W. Chang. Routing for manufacturability and reliability. *IEEE Circuits and Systems Magazine*, 9(3):20–31, 2009.

[CCM+20] G. Chen, W. Chen, Y. Ma, H. Yang, B. Yu. Damo: Deep agile mask optimization for full chip scale. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[CCS+06] T. F. Chan, J. Cong, J. R. Shinnerl, K. Sze, M. Xie. mpl6: Enhanced multilevel mixed-size placement. In *Proceedings of the 2006 international symposium on Physical design*, pages 212–214, 2006.

[CCWW00] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, S.-W. Wu. B*-trees: A new representation for non-slicing floorplans. In *Proceedings of the 37th annual design automation conference*, pages 458–463, 2000.

[Cha24] Y.-W. Chang. Physical design challenges in modern heterogeneous integration. In *Proceedings of the 2024 International Symposium on Physical Design*, pages 125–134, 2024.

[Che25] B. Chen. Application of photolithography in integrated circuits. *Applied and Computational Engineering*, 126:33–38, 2025.

[Chu04] C. Chu. Flute: Fast lookup table based technique for rsmt construction and wirelength estimation, 2004.

[CJH+07] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, Y.-W. Chang. Ntuplace3: An analytical placer for large-scale mixed-size designs. In *Modern Circuit Placement: Best Practices and Results*, pages 289–309. Springer, 2007.

[CJH+08] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, Y.-W. Chang. Ntuplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(7):1228–1240, 2008.

[CKKW18] C.-K. Cheng, A. B. Kahng, I. Kang, L. Wang. Replace: Advancing solution quality and routability validation in global placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(9):1717–1730, 2018.

[CKZ+23] J. Chen, J. Kuang, G. Zhao, D. J.-H. Huang, E. F. Y. Young. Pros 2.0: A plug-in for routability optimization and routed wirelength estimation using deep learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(1):164–177, 2023. doi:10.1109/TCAD.2022.3168259.

[CLC08] T.-C. Chen, G.-W. Liao, Y.-W. Chang. Predictive formulae for opc with applications to lithography-friendly routing. In *Proceedings of the 45th annual Design Automation Conference*, pages 510–515, 2008.

[CLC10] T.-C. Chen, G.-W. Liao, Y.-W. Chang. Predictive formulae for opc with applications to lithography-friendly routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(1):40–50, 2010.

[CLG+10] Y.-J. Chang, Y.-T. Lee, J.-R. Gao, P.-C. Wu, T.-C. Wang. Nthuroute 2.0: a robust global router for modern designs. *IEEE Trans-*

actions on Computer-Aided Design of Integrated Circuits and Systems, 29(12):1931–1944, 2010.

[CLL+22] R. Cheng, X. Lyu, Y. Li, J. Ye, J. Hao, J. Yan. The policy-gradient placement and generative routing neural networks for chip design. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 26350–26362. Curran Associates, Inc., 2022. doi:10.52202/068431-1911.

[CLYP09] M. Cho, K. Lu, K. Yuan, D. Z. Pan. Boxrouter 2.0: A hybrid and robust global router with layer assignment for routability. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 14(2):1–21, 2009.

[CP06] M. Cho, D. Z. Pan. Boxrouter: A new global router based on box expansion and progressive ilp. In *Proceedings of the 43rd annual Design Automation Conference*, pages 373–378, 2006.

[CSD+24] Y. Chen, M. Su, H. Ding, S. Weng, Z. Lin, X. Bai. High-correlation 3d routability estimation for congestion-guided global routing. *The Journal of Supercomputing*, 80(3):3114–3141, 2024.

[CW07] C. Chu, Y.-C. Wong. Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(1):70–83, 2007.

[CXX23] Y. Chen, Y. Xu, N. Xu. Scalable multiple patterning layout decomposition implemented by a distribution evolutionary algorithm. *arXiv preprint arXiv:2304.04207*, 2023.

[CYY23] G. Chen, H. Yang, B. Yu. Gpu-accelerated matrix cover algorithm for multiple patterning layout decomposition. In *DTCO and Computational Patterning II*, volume 12495, pages 376–383. SPIE, 2023.

[CZY+19] R. Chen, W. Zhong, H. Yang, H. Geng, X. Zeng, B. Yu. Faster region-based hotspot detection. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[DAL15] S. Dong, J. Ao, F. Luo. Delay-driven and antenna-aware layer assignment in global routing under multitier interconnect structure. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(5):740–752, 2015.

[DB20] V. P. Dwivedi, X. Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

[DBH18] P. De Bisschop, E. Hendrickx. Stochastic effects in euv lithography. In *Extreme Ultraviolet (EUV) Lithography IX*, volume 10583, pages 350–366. SPIE, 2018.

[DYXZ21] H. Du, Z. Yan, Q. Xiang, Q. Zhan. Vulcan: Solving the steiner tree problem with graph neural networks and deep reinforcement learning. *arXiv preprint arXiv:2111.10810*, 2021.

[FC12] S.-Y. Fang, Y.-W. Chang. Simultaneous flare level and flare variation minimization with dummification in euvl. In *Proceedings of the 49th Annual Design Automation Conference*, pages 1179–1184, 2012.

[FCC12a] S.-Y. Fang, Y.-W. Chang, W.-Y. Chen. A novel layout decomposition algorithm for triple patterning lithography. In *Proceedings of the 49th Annual Design Automation Conference*, pages 1185–1190, 2012.

[FCC12b] S.-Y. Fang, S.-Y. Chen, Y.-W. Chang. Native-conflict and stitch-aware wire perturbation for double patterning technology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(5):703–716, 2012.

[FCL+22] J. Feng, Y. Chen, F. Li, A. Sarkar, M. Zhang. How powerful are k-hop message passing graph neural networks. *Advances in Neural Information Processing Systems*, 35:4776–4790, 2022.

[FLK+20] F. V. Fomin, D. Lokshtanov, S. Kolay, F. Panolan, S. Saurabh. Subexponential algorithms for rectilinear steiner tree and arborescence problems. *ACM Transactions on Algorithms (TALG)*, 16(2):1–37, 2020.

[FM82] C. M. Fiduccia, R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, DAC '82, page 175–181. IEEE Press, 1982.

[Fre12] Free Software Foundation. Gnu linear programming kit (glpk). https://www.gnu.org/software/glpk/, 2012. Accessed: 2020-08-01.

[FS12] D. Forte, A. Srivastava. On improving the uniqueness of silicon-based physically unclonable functions via optical proximity correction. In *Proceedings of the 49th Annual Design Automation Conference*, pages 96–105, 2012.

[FS17] FEMTO-ST. Electron beam lithography - raith e_line, 2017. URL https://www.femto-st.fr/en/Technology-center/MIMENTO-Technology-Centre/Available-resources/Nanotechnologies/electron-beam-lithography-raith-e-line. Last modified: 04 Oct 2017.

[Fuj22] T. Fujimori. Recent status of the stochastic issues of photoresist materials in euv lithography. In *2022 International Workshop on Advanced Patterning Solutions (IWAPS)*, pages 1–3. IEEE, 2022.

[GG16] Y. Gal, Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[GHM+24] L. Gottesbüren, T. Heuer, N. Maas, P. Sanders, S. Schlag. Scalable high-quality hypergraph partitioning. *ACM Transactions on Algorithms*, 20(1):1–54, 2024.

[GJ77] M. R. Garey, D. S. Johnson. The rectilinear steiner tree problem is np-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

[GJ+90] M. R. Garey, D. S. Johnson, others. A guide to the theory of np-completeness. *Computers and intractability*, pages 37–79, 1990.

[GJLP20] J. Gu, Z. Jiang, Y. Lin, D. Z. Pan. Dreamplace 3.0: Multi-electrostatics based robust vlsi placement with region constraints. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[GPAM+14] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[GRDY+12] R. Gronheid, P. A. Rincon Delgadillo, T. R. Younkin, I. Pollentier, M. Somervell, J. S. Hooge, K. Nafus, P. F. Nealey. Frequency multiplication of lamellar phase block copolymers with graphoepitaxy directed self-assembly sensitivity to prepattern. *Journal of Micro/Nanolithography, MEMS, and MOEMS*, 11(3):031303–031303, 2012.

[GYZ+20] H. Geng, H. Yang, L. Zhang, J. Miao, F. Yang, X. Zeng, B. Yu. Hotspot detection via attention-based deep layout metric learning. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–8, 2020.

[Had75] F. Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal on Computing*, 4(3):221–225, 1975.

[Ham25] S. Hambric. High sensitivity accelerometers for monitoring semiconductor manufacturing. In *INTER-NOISE and NOISE-CON Congress and Conference Proceedings*, volume 272, pages 723–732. Institute of Noise Control Engineering, 2025.

[Han66] M. Hanan. On steiner's problem with rectilinear distance. *SIAM Journal on Applied mathematics*, 14(2):255–265, 1966.

[Har82] J. Hartmanis. Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson). *Siam Review*, 24(1):90, 1982.

[HF20] C.-H. Hsu, S.-Y. Fang. Stitch-aware routing considering smart boundary for multiple e-beam lithography. In *2020 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pages 1–4. IEEE, 2020.

[HK92] L. Hagen, A. B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE transactions on computer-aided design of integrated circuits and systems*, 11(9):1074–1085, 1992.

[HL18] R. M. M. Hasan, X. Luo. Promising lithography techniques for next-generation logic devices. *Nanomanufacturing and Metrology*, 1(2):67–81, 2018.

[HLC10] S.-L. Huang, C.-W. Lin, Y.-W. Chang. Efficient provably good opc modeling and its applications to interconnect optimization. In *2010 IEEE International Conference on Computer Design*, pages 336–341. IEEE, 2010.

[HLG+19] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, J. Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

[HMZ+20] Z. He, Y. Ma, L. Zhang, P. Liao, N. Wong, B. Yu, M. D. Wong. Learn to floorplan through acquisition of effective local search heuristics. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pages 324–331. IEEE, 2020.

[HWL+23] W. Huang, J. Wu, W. Li, G. Chen, C. Chu, C. Li, Y. Zhu, H. Yang, Y. Chao. Fabrication of silicon nanowires by metal-assisted chemical etching combined with micro-vibration. *Materials*, 16(15):5483, 2023.

[HYL17] W. Hamilton, Z. Ying, J. Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[IZZE17] P. Isola, J.-Y. Zhu, T. Zhou, A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[JLM+20] B. Jiang, L. Liu, Y. Ma, H. Zhang, B. Yu, E. F. Young. Neural-ilt: Migrating ilt to neural networks for mask printability and complexity co-optimization. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[JLM+21] B. Jiang, L. Liu, Y. Ma, B. Yu, E. F. Young. Neural-ilt 2.0: Migrating ilt to domain-specific and multitask-enabled neural network. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(8):2671–2684, 2021.

[JLY+22] M. Jiang, D. Liang, B. Yan, L. Li, M. Gao, J. Huang, A. Lan, J. Shi. Resolution improvement review for the immersion lithography. In *2022 International Workshop on Advanced Patterning Solutions (IWAPS)*, pages 1–4. IEEE, 2022.

[JO17] S. R. Jain, K. Okabe. Training a fully convolutional neural network to route integrated circuits. *arXiv preprint arXiv:1706.08948*, 2017.

[JWWZ18] D. Juhl, D. M. Warme, P. Winter, M. Zachariasen. The geosteiner software package for computing steiner trees in the plane: an updated computational study. *Mathematical Programming Computation*, 10(4):487–532, 2018.

[JYY+20] Y. Jiang, F. Yang, B. Yu, D. Zhou, X. Zeng. Efficient layout hotspot detection via binarized residual neural network ensemble. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(7):1476–1488, 2020.

[Kah22] A. B. Kahng. Machine learning for cad/eda: The road ahead. *IEEE Design & Test*, 40(1):8–16, 2022.

[KC24] A. M. Kulkarni, A. Chopde. Physical design: Methodologies and developments. *arXiv preprint arXiv:2409.04726*, 2024.

[KG17] A. Kendall, Y. Gal. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, 30, 2017.

[KG21] S. Kumar, A. Gupta. *Integrated circuit fabrication.* CRC Press, 2021.

[KGTC23] A. Kumar, S. N. Ghosh, S. Talukder, D. Chopra. Lithography and 3d fabrication processes: a review. *ES Materials & Manufacturing*, 23:974, 2023.

[Kip16] T. Kipf. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[KL70] B. W. Kernighan, S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.

[KLMH11] A. B. Kahng, J. Lienig, I. L. Markov, J. Hu. *VLSI physical design: from graph partitioning to timing closure*, volume 312. Springer, 2011.

[KODG24] M. H. Khan, B. Onal, E. Dogan, M. R. Guthaus. Vlsi hypergraph partitioning with deep learning. *arXiv preprint arXiv:2409.01387*, 2024.

[Koo21] B. Koonce. Resnet 50. In *Convolutional neural networks with swift for tensorflow: image recognition and dataset categorization*, pages 63–72. Springer, 2021.

[KRW05] A. B. Kahng, S. Reda, Q. Wang. Aplace: A general analytic placement framework. In *Proceedings of the 2005 international symposium on Physical design*, pages 233–235, 2005.

[KSKM18] B. Kar, S. Sur-Kolay, C. Mandal. Stairoute: Early global routing using monotone staircases for congestion reduction. *arXiv preprint arXiv:1810.10412*, 2018.

[LC14] C.-Y. Liu, Y.-W. Chang. Simultaneous euv flare-and cmp-aware placement. In *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, pages 249–255. IEEE, 2014.

[LCC+15] J. Lu, P. Chen, C.-C. Chang, L. Sha, D. J.-H. Huang, C.-C. Teng, C.-K. Cheng. eplace: Electrostatics-based placement using fast fourier transform and nesterov's method. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 20(2):1–34, 2015.

[LCY21] J. Liu, G. Chen, E. F. Young. Rest: Constructing rectilinear steiner minimum tree via reinforcement learning. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 1135–1140. IEEE, 2021.

[LCZ22] L. Li, Y. Cai, Q. Zhou. A survey on machine learning-based routing for vlsi physical design. *Integration*, 86:51–56, 2022.

[LDL+19] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, D. Z. Pan. Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[Lev22] H. J. Levinson. High-na euv lithography: current status and outlook for the future. *Japanese Journal of Applied Physics*, 61(SD):SD0803, 2022.

[LFC15] I.-J. Liu, S.-Y. Fang, Y.-W. Chang. Stitch-aware routing for multiple e-beam lithography. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(3):471–482, 2015.

[LGG+17] T.-Y. Lin, P. Goyal, R. Girshick, K. He, P. Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[LGG+23] P. Liao, D. Guo, Z. Guo, S. Liu, Y. Lin, B. Yu. Dreamplace 4.0: Timing-driven placement with momentum-based net weighting and lagrangian-based refinement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(10):3374–3387, 2023.

[LHW+24] H. Li, Y. Huo, Y. Wang, X. Yang, M. Hao, X. Wang. A lightweight inception boosted u-net neural network for routability prediction. In *2024 2nd International Symposium of Electronics Design Automation (ISEDA)*, pages 648–653, 2024. doi:10.1109/ISEDA62518.2024.10617987.

[Lin12] B. J. Lin. Future of multiple-e-beam direct-write systems. *Journal of Micro/Nanolithography, MEMS, and MOEMS*, 11(3):033011–033011, 2012.

[LK19] S.-E. D. Lin, D. H. Kim. Construction of all rectilinear steiner minimum trees on the hanan grid and its applications to vlsi design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(6):1165–1176, 2019.

[LLH+24] Z. Lin, G. Liu, X. Huang, Y. Lin, J. Zhang, W.-H. Liu, T.-C. Wang. A unified deep reinforcement learning approach for constructing rectilinear and octilinear steiner minimum tree. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[LLZ+24] H. Liu, P. Liao, M. Zou, B. Pang, X. Li, M. Yuan, T.-Y. Ho, B. Yu. Layout decomposition via boolean satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[LMMS17] L. Lavagno, I. L. Markov, G. Martin, L. K. Scheffer. *Electronic design automation for IC implementation, circuit design, and process technology*. CRC Press, 2017.

[LMS+20] W. Li, Y. Ma, Q. Sun, L. Zhang, Y. Lin, I. H.-R. Jiang, B. Yu, D. Z. Pan. Openmpl: An open-source layout decomposer. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(11):2331–2344, 2020.

[LPB17] B. Lakshminarayanan, A. Pritzel, C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

[LPRK20] Y. Lin, D. Z. Pan, H. Ren, B. Khailany. Dreamplace 2.0: Open-source gpu-accelerated global and detailed placement for large-scale vlsi designs. In *2020 China Semiconductor Technology International Conference (CSTIC)*, pages 1–4. IEEE, 2020.

[LPZ+20] Y.-C. Lu, S. S. K. Pentapati, L. Zhu, K. Samadi, S. K. Lim. Tp-gnn: A graph neural network framework for tier partitioning in monolithic 3d ics. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[LSL+21] S. Liu, Q. Sun, P. Liao, Y. Lin, B. Yu. Global placement with deep learning-enabled explicit routability optimization. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1821–1824. IEEE, 2021.

[LTBZ15] Y. Li, D. Tarlow, M. Brockschmidt, R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[LW09] T.-H. Lee, T.-C. Wang. Robust layer assignment for via optimization in multi-layer global routing. In *Proceedings of the 2009 international symposium on Physical design*, pages 159–166, 2009.

[LXJ+20] R. Liang, Z. Xie, J. Jung, V. Chauha, Y. Chen, J. Hu, H. Xiang, G.-J. Nam. Routing-free crosstalk prediction. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[LYZ+17] Y. Lin, B. Yu, Y. Zou, Z. Li, C. J. Alpert, D. Z. Pan. Stitch aware detailed placement for multiple e-beam lithography. *Integration*, 58:47–54, 2017.

[LZD+24] R. Liu, Z. Zeng, S. Ding, J. Sui, X. Li, D. Bu. Neuralsteiner: Learning steiner tree for overflow-avoiding global routing in chip design. *Advances in Neural Information Processing Systems*, 37:127346–127368, 2024.

[Mac08] C. Mack. *Fundamental principles of optical lithography: the science of microfabrication*. John Wiley & Sons, 2008.

[MGY+21] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nova, others. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

[MHK12] I. L. Markov, J. Hu, M.-C. Kim. Progress and challenges in vlsi placement research. In *Proceedings of the International Conference on Computer-Aided Design*, pages 275–282, 2012.

[MHR06] P. Miettinen, M. Honkala, J. Roos. *Using METIS and hMETIS algorithms in circuit partitioning*. Helsinki University of Technology, 2006.

[MO14] M. Mirza, S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[Mof08] M. D. Moffitt. Maizerouter: Engineering an effective global router. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(11):2017–2026, 2008.

[Mor12] W. M. Moreau. *Semiconductor lithography: principles, practices, and materials*. Springer Science & Business Media, 2012.

[Nei21] M. Neisser. International roadmap for devices and systems lithography roadmap. *Journal of Micro/Nanopatterning, Materials, and Metrology*, 20(4):044601–044601, 2021.

[oCADI21] I. C. on Computer-Aided Design (ICCD). Faraday benchmarks. http://vlsicad.eecs.umich.edu/BK/ICCAD04bench/FARADAYICCAD04Bench.tar.gz, 2021. Accessed: 23 March 2021.

[PBM00] N. W. Parker, A. D. Brodie, J. H. McCoy. High-throughput ngl electron-beam direct-write lithography system. In *Emerging Lithographic Technologies IV*, volume 3997, pages 713–720. SPIE, 2000.

[PBSK17] S. Paul, P. Banerjee, S. Sur-Kolay. Post-layout perturbation towards stitch friendly layout for multiple e-beam lithography. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 411–414. IEEE, 2017.

[PBSK19] S. Paul, P. Banerjee, S. Sur-Kolay. Minimization of flare in euvl by simultaneous wire segment perturbation and dummification. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 212–217. IEEE, 2019.

[PBSK21] S. Paul, P. Banerjee, S. Sur-Kolay. A study on flare minimisation in euv lithography by post-layout re-allocation of wire segments. *IET Circuits, Devices & Systems*, 15(4):310–329, 2021.

[PLL+24] H. Pan, C. Lan, Y. Liu, Z. Wang, L. Shang, X. Zeng, F. Yang, K. Zhu. Physically aware synthesis revisited: guiding technology mapping with primitive logic gate placement. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2024.

[PMBSK23] S. Paul, T. Mukherjee, P. Banerjee, S. Sur-Kolay. Concurrent steiner tree selection for global routing with euvl flare reduction. *Integration*, 92:66–76, 2023.

[PSDV+18] E. Perez, F. Strub, H. De Vries, V. Dumoulin, A. Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[PXZC12] M. Pan, Y. Xu, Y. Zhang, C. Chu. Fastroute: An efficient and high-quality global router. *VLSI Design*, 2012(1):608362, 2012.

[PYC+12] S. Park, H. Youn, N. Chung, J. Maeng, S. Lee, J. Ku, X. Xie, S. Lan, M. Feng, V. Vellanki, others. Model based opc for implant layer patterning considering wafer topography proximity (w3d) effects. In *Optical Microlithography XXV*, volume 8326, pages 237–243. SPIE, 2012.

[QXZ+23] Y. Qiu, Y. Xing, X. Zheng, P. Gao, S. Cai, X. Xiong. Progress of placement optimization for accelerating vlsi physical design. *Electronics*, 12(2):337, 2023.

[RFB15] O. Ronneberger, P. Fischer, T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[RSV11] J. Ramanujam, D. Shiri, A. Verma. Silicon nanowire growth and properties: a review. *Materials Express*, 1(2):105–126, 2011.

[SAF25] H. Sajadinia, A. Aghdaei, Z. Feng. Shypar: A spectral coarsening approach to hypergraph partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025.

[Sau23] S. Saurabh. *Introduction to VLSI design flow*. Cambridge University Press, 2023.

[SBSK22] K. Saha, P. Banerjee, S. Sur-Kolay. Stitch-avoiding detailed routing for multiple e-beam lithography. In *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6. IEEE, 2022.

[SBSK25] K. Saha, P. Banerjee, S. Sur-Kolay. Constructing rectilinear steiner minimum tree with conditional generative adversarial network. In *2025 38th International Conference on VLSI Design and 2024 23rd International Conference on Embedded Systems (VLSID)*, pages 351–356. IEEE, 2025.

[SC15] Y.-H. Su, Y.-W. Chang. Nanowire-aware routing considering high cut mask complexity. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, 2015.

[She99] N. A. Sherwani. *Algorithms for VLSI physical design automation*. Springer, 1999.

[SMM08] M. Sugihara, Y. Matsunaga, K. Murakami. Character projection mask set optimization for enhancing throughput of mcc projection systems. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 91(12):3451–3460, 2008.

[SPBSK22] K. Saha, S. Paul, P. Banerjee, S. Sur-Kolay. Stitch-avoiding global routing for multiple e-beam lithography. In *2022 35th International Conference on VLSI Design and 2022 21st International Conference on Embedded Systems (VLSID)*, pages 138–143. IEEE, 2022.

[SRM+22] E. Sharma, R. Rathi, J. Misharwal, B. Sinhmar, S. Kumari, J. Dalal, A. Kumar. Evolution in lithography techniques: microlithography to nanolithography. *Nanomaterials*, 12(16):2754, 2022.

[SW17] Y. Saatci, A. G. Wilson. Bayesian gan. *Advances in neural information processing systems*, 30, 2017.

[SWT05] F. M. Schellenberg, J. Word, O. Toublan. Layout compensation for euv flare. In *Emerging Lithographic Technologies IX*, volume 5751, pages 320–329. SPIE, 2005.

[TLA+10] T. Taghavi, Z. Li, C. Alpert, G.-J. Nam, A. Huber, S. Ramji. New placement prediction and mitigation techniques for local routing congestion. In *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 621–624. IEEE, 2010.

[TLCX20] H. Tang, G. Liu, X. Chen, N. Xiong. A survey on steiner tree construction and global routing for vlsi design. *IEEE Access*, 8:68593–68622, 2020.

[VCC+17] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[VNA+07] N. Viswanathan, G.-J. Nam, C. J. Alpert, P. Villarrubia, H. Ren, C. Chu. Rql: Global placement via relaxed quadratic spreading and linearization. In *Proceedings of the 44th annual Design Automation Conference*, pages 453–458, 2007.

[VPC07] N. Viswanathan, M. Pan, C. Chu. Fastplace: An efficient multilevel force-directed placement algorithm. In *Modern Circuit Placement: Best Practices and Results*, pages 193–228. Springer, 2007.

[VSP+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[WDL09] T.-H. Wu, A. Davoodi, J. T. Linderoth. Grip: Scalable 3d global routing using integer programming. In *Proceedings of the 46th Annual Design Automation Conference*, pages 320–325, 2009.

[WH15] N. H. Weste, D. Harris. *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.

[WJK+25] C. Wang, W. Ji, M. Kou, Z. Chen, F. Li, N. Zhu, H. Yao. Mr. tpl: A method for multi-pin net router in triple patterning lithography. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2025.

[WLZ+18] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, B. Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8798–8807, 2018.

[Wol08] W. Wolf. *Modern VLSI design: IP-based design*. Prentice Hall PTR, 2008.

[XHF+18] Z. Xie, Y.-H. Huang, G.-Q. Fang, H. Ren, S.-Y. Fang, Y. Chen, J. Hu. Routenet: Routability prediction for mixed-size designs using convolutional neural network. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2018.

[XLT+18] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, S. Jegelka. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*, pages 5453–5462. pmlr, 2018.

[XR05] Z. Xiu, R. A. Rutenbar. Timing-driven placement by grid-warping. In *Proceedings of the 42nd annual Design Automation Conference*, pages 585–591, 2005.

[XZC09] Y. Xu, Y. Zhang, C. Chu. Fastroute 4.0: Global router with efficient via minimization. In *2009 Asia and South Pacific Design Automation Conference*, pages 576–581. IEEE, 2009.

[YALP19] W. Ye, M. B. Alawieh, Y. Lin, D. Z. Pan. Lithogan: End-to-end lithography modeling with generative adversarial networks. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[YC23] C. Yang, K. R. Chandran. A critical review of silicon nanowire electrodes and their energy storage capacities in li-ion cells. *RSC advances*, 13(6):3947–3957, 2023.

[YLLP+13] B. Yu, Y.-H. Lin, G. Luk-Pat, D. Ding, K. Lucas, D. Z. Pan. A high-performance triple patterning layout decomposer with balanced density. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 163–169. IEEE, 2013.

[YLM+18] H. Yang, S. Li, Y. Ma, B. Yu, E. F. Young. Gan-opc: Mask optimization with lithography-guided generative adversarial nets. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[Yos84] T. Yoshimura. An efficient channel router. In *21st Design Automation Conference Proceedings*, pages 38–44. IEEE, 1984.

[YP14] B. Yu, D. Z. Pan. Layout decomposition for quadruple patterning lithography and beyond. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6, 2014.

[YPG+19] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, B. Yu. Detecting multi-layer layout hotspots with adaptive squish patterns. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pages 299–304, 2019.

[YPMZ15] B. Yu, D. Z. Pan, T. Matsunawa, X. Zeng. Machine learning and pattern matching in physical design. In *The 20th Asia and South Pacific design automation conference*, pages 286–293. IEEE, 2015.

[YRGP15] B. Yu, S. Roy, J.-R. Gao, D. Z. Pan. Triple patterning lithography layout decomposition using end-cutting. *Journal of Micro/Nanolithography, MEMS, and MOEMS*, 14(1):011002–011002, 2015.

[YSJ+19] K. Yang, K. Swanson, W. Jin, C. Coley, P. Eiden, H. Gao, A. Guzman-Perez, T. Hopper, B. Kelley, M. Mathea, others. Analyzing learned molecular representations for property prediction. *Journal of chemical information and modeling*, 59(8):3370–3388, 2019.

[YYDP15] B. Yu, K. Yuan, D. Ding, D. Z. Pan. Layout decomposition for triple patterning lithography. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(3):433–446, 2015.

[YYG+16] B. Yu, K. Yuan, J.-R. Gao, S. Hu, D. Z. Pan. Ebl overlapping aware stencil planning for mcc system. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 21(3):1–24, 2016.

[YYGP13] B. Yu, K. Yuan, J.-R. Gao, D. Z. Pan. E-blow: E-beam lithography overlapping aware stencil planning for mcc system. In *Proceedings of the 50th Annual Design Automation Conference*, pages 1–7, 2013.

[YZ19] C. Yu, Z. Zhang. Painting on placement: Forecasting routing congestion using conditional generative adversarial nets. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[YZM+21] H. Yang, W. Zhong, Y. Ma, H. Geng, R. Chen, W. Chen, B. Yu. Vlsi mask optimization: From shallow to deep learning. *Integration*, 77:96–103, 2021.

[ZC12] Y. Zhang, C. Chu. Regularroute: An efficient detailed router applying regular routing patterns. *IEEE transactions on very large scale integration (VLSI) systems*, 21(9):1655–1668, 2012.

[ZHW+11] C. Zuniga, M. Habib, J. Word, G. F. Lorusso, E. Hendrickx, B. Baylav, R. Chalasani, M. Lam. Euv flare and proximity modeling and model-based correction. In *Extreme Ultraviolet (EUV) Lithography II*, volume 7969, pages 262–274. SPIE, 2011.

[ZMCL17] C. Zhang, X. Miao, K. D. Chabak, X. Li. A review of iii–v planar nanowire arrays: Selective lateral vls epitaxy and 3d transistors. *Journal of Physics D: Applied Physics*, 50(39):393001, 2017.