

© 2012 Qiang Ma

ROUTING ALGORITHMS FOR ELECTRONIC DESIGN AUTOMATION

BY

QIANG MA

DISSERTATION

Submitted in partial fulfillment of the requirements  
for the degree of Doctor of Philosophy in Electrical and Computer Engineering  
in the Graduate College of the  
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Doctoral Committee:

Professor Martin D. F. Wong, Chair, Director of Research  
Associate Professor Deming Chen  
Professor Robin A. Rutenbar  
Assistant Professor Shobha Vasudevan

# ABSTRACT

In electronic design automation (EDA), routing is one of the most important tasks for both printed circuit boards (PCB) and integration circuits (IC). After placement, which determines the location of each component on a PCB or element of an IC, the routing step adds wires needed to properly connect the placed components/elements while obeying all the design rules. The quality of routing has a great impact on the performance of the board/chip. In this dissertation, we study modern PCB routing and the emerging problems in IC routing.

The high pin density and large net count of modern PCBs make manual design the board an extremely time-consuming and error-prone task. In addition, the bus structure routing style and the planar routing constraint further increase the design complexity. The number of layers used also needs to be minimized in order to reduce the fabrication cost. To the best of our knowledge, no mature commercial automated router can handle these problems well. In this dissertation, both bus-based and net-based PCB routing problems are addressed. We first introduce the Rectangle Escape Problem (REP), which is motivated by bus-based escape routing within one component on a PCB. This problem is basically to determine an escape direction for each bus within the component such that the resultant maximum density (a good indicator of the number of layers needed) is minimized. We prove that REP is NP-complete, and show that it can be formulated as an integer linear program (ILP). A provably good approximation algorithm for the REP is developed by applying linear programming (LP) relaxation and a special rounding technique to the ILP. We further study the optimal layer assignment of a set of buses connecting two components on a PCB. This is a theoretically hard problem and we propose a branch-and-bound based algorithm that optimally solves it. Our algorithm is guaranteed to produce a feasible layer assignment of the buses with a minimum number of layers.

We apply our algorithms on industrial data and the experimental results validate our approach. Net-based PCB routing is also studied. We propose an underlying routing graph which correctly models the routing resources of the pin grids on board. We then build a Negotiated Congestion based Escape Router (NCER) by applying the negotiated congestion routing scheme on the constructed routing graph. We compare the performance of NCER with that of the Cadence PCB router Allegro on industrial test cases, and experimental results show that the two routers have comparable routability but complementary behaviors. Therefore, by using NCER as a supplement to Allegro, we can solve a broader range of net-based PCB routing problems.

The emerging problems in IC routing are also studied. Conventional CMOS devices face an increasing number of challenges as their feature sizes scale down. Graphene nanoribbon (GNR) based devices are shown to be a promising replacement of traditional CMOS at future technology nodes. However, all previous works on GNRs focus at the device level. In order to integrate these devices into electronic systems, routing becomes a key issue. We study the GNR routing problem for the first time. We formulate the GNR routing problem as a minimum hybrid-cost shortest path problem on triangular mesh (“hybrid” means that we need to consider both the length and the bending of the routing path). We show that by graph expansion, this minimum hybrid-cost shortest path problem can be solved by applying the conventional shortest path algorithm on the expanded graph. Experimental results show that our GNR routing algorithm effectively handles the hybrid cost. We then study the related routing reliability problem. In practice, the GNR wire segments can have a connection defective rate. Particularly, each wire segment has a survival probability, and thus has a chance to fail, making traditional routing very unreliable. We introduce the routing reliability problem and propose an algorithm flow to solve it. Given an  $s$ - $t$  routing path on a routing graph, we try to reinforce the reliability of the routing path by adding redundant wiring segments in such a way that its survival probability is maximized with a reasonable overhead of routing resources. Our proposed algorithm flow is two-fold: (1) generation of candidate redundancy segment via min-cost max-flow; (2) optimal selection among the candidates by dynamic programming. The results of extensive experiments confirm the effectiveness and efficiency of our approach.

Another emerging problem in IC routing that we study is triple patterning



lithography (TPL) aware routing. As technology continues to scale to 14nm node, double patterning lithography (DPL) is pushed to near its limit. TPL is a considerable and natural extension along the paradigm of DPL. With an extra mask to accommodate the features, TPL can be used to eliminate the unresolvable conflicts and minimize the number of stitches, which are pervasive in DPL process, and thus smoothen the layout decomposition step. Considering TPL during the routing stage explores a larger solution space and can further improve the layout decomposability. We propose the first triple patterning aware detailed routing scheme, and compare its performance with the double patterning version in 14nm node. Experimental results show that, using TPL, the conflicts can be resolved much more easily and the stitches can be significantly reduced in contrast to DPL.

*To world peace.*

# ACKNOWLEDGMENTS

I owe my deepest gratitude to my adviser, Prof. Martin D. F. Wong. He has been constantly helping me in many aspects: from establishing research topics to presenting research results. This dissertation would not have been possible without his patient guidance, inspiring discussions and abundant encouragement.

Besides my adviser, I would like to thank the rest of my Doctoral Committee: Prof. Deming Chen, Prof. Rob A. Rutenbar and Prof. Shobha Vasudevan for their insightful comments and constructive suggestions. Their invaluable opinions have significantly improved the quality of this dissertation.

I also want to thank Prof. Evangeline F. Y. Young, my master's degree adviser, for the inspiring discussions we had and the valuable suggestions she provided on my research topics during the summer of 2010 at Chinese University of Hong Kong. Moreover, I want to thank Dr. Tan Yan for his recommendation for the internship opportunity at Synopsys Inc. in 2011, where I had my first industrial experience that turned out to be of great help to my research and job hunting.

All the members of Prof. Wong's research team who have overlapped with me made my life in UIUC very enjoyable. I would like to thank Dr. Hui Kong, Dr. Tan Yan, Dr. Lijuan Luo, Dr. Hongbo Zhang, Ms. Pei-Ci Wu, Mr. Yuelin Du, Ms. Ting Yu, Ms. Leslie Hwang, Mr. Zigang Xiao and Mr. Haitong Tian for all the stimulating discussions and the seamless collaborations we had, as well as their help in my study and life.

Lastly, I would like to thank my family for their endless love and support. I want to thank my parents for raising me up and putting all their efforts into educating me. Words cannot express my thanks to my wife, Xuan Wang. I really appreciate her love, support, and encouragement throughout my Ph.D. study.

# TABLE OF CONTENTS

LIST OF TABLES . . . . .	ix
LIST OF FIGURES . . . . .	x
LIST OF ABBREVIATIONS . . . . .	xiv
CHAPTER 1 INTRODUCTION . . . . .	1
1.1 Background and Motivation . . . . .	1
1.2 Overview of this Dissertation . . . . .	3
CHAPTER 2 BUS-BASED ESCAPE ROUTING ON PCB . . . . .	7
2.1 Introduction . . . . .	7
2.2 Problem Definition . . . . .	11
2.3 Proof of NP-Completeness . . . . .	11
2.4 The Algorithm . . . . .	16
2.5 Extensions . . . . .	24
2.6 Experimental Results . . . . .	29
2.7 Concluding Remarks . . . . .	32
CHAPTER 3 NET-BASED ESCAPE ROUTING ON PCB . . . . .	34
3.1 Introduction . . . . .	34
3.2 Problem Formulation . . . . .	36
3.3 Underlying Routing Graph . . . . .	37
3.4 Negotiated Congestion . . . . .	39
3.5 Application on Bus Untangling . . . . .	41
3.6 Experimental Results . . . . .	42
3.7 Conclusion . . . . .	46
CHAPTER 4 LAYER ASSIGNMENT OF PCB ROUTING . . . . .	47
4.1 Introduction . . . . .	47
4.2 Problem Formulation . . . . .	51
4.3 Optimal Layer Assignment . . . . .	53
4.4 Experimental Results . . . . .	64
4.5 Concluding Remarks . . . . .	66

CHAPTER 5	ROUTING WITH GRAPHENE NANORIBBONS . . .	67
5.1	Introduction . . . . .	67
5.2	Background . . . . .	69
5.3	GNR Routing Problem . . . . .	71
5.4	Solving the MHCP Problem . . . . .	75
5.5	Negotiated Congestion Routing Scheme . . . . .	85
5.6	Experimental Result . . . . .	86
5.7	Conclusion . . . . .	87
CHAPTER 6	THE ROUTING RELIABILITY PROBLEM . . . . .	89
6.1	Introduction . . . . .	89
6.2	Preliminaries . . . . .	91
6.3	Problem Definition . . . . .	94
6.4	Algorithm Flow for the Routing Reliability Problem . . . . .	94
6.5	Experimental Results . . . . .	103
6.6	Concluding Remarks . . . . .	106
CHAPTER 7	TRIPLE PATTERNING AWARE ROUTING . . . . .	108
7.1	Introduction . . . . .	108
7.2	Problem Formulation . . . . .	111
7.3	Routing a Single Net . . . . .	112
7.4	Overall Routing Scheme . . . . .	118
7.5	Experimental Results . . . . .	121
7.6	Concluding Remarks . . . . .	124
CHAPTER 8	CONCLUSIONS . . . . .	125
REFERENCES	. . . . .	128

# LIST OF TABLES

2.1	Comparison with ILP and greedy approach . . . . .	30
2.2	Experimental results on weighted test cases . . . . .	32
3.1	NCER vs. Cadence Allegro on industrial simultaneous es- cape routing test cases . . . . .	45
4.1	Runtime of ILP on MIS and MC . . . . .	66
4.2	Results of our branch-and-bound algorithm . . . . .	66
5.1	Edge costs of the graph in Figure 5.7 . . . . .	76
5.2	Edge costs of the graph in Figure 5.8 . . . . .	76
5.3	Experimental Results . . . . .	86
6.1	Experimental results on general graphs with randomized edge costs and survival probabilities . . . . .	104
7.1	Comparison of TPL and DPL in 14nm technology . . . . .	122
7.2	Comparison of TPL and DPL in resolving conflicts . . . . .	124

# LIST OF FIGURES

2.1	The projection rectangle is obtained by projecting the bounding box of the pin cluster of the bus to one of the boundaries of the component. . . . .	7
2.2	The four projection rectangles of a bus and its escape routing to the right boundary of the component. . . . .	8
2.3	The routes of two buses conflict. . . . .	8
2.4	Illustration of Rectangle Escape Problem: (a) input rectangles; (b) an escape solution with maximum density equal to 2; (c) an escape solution with maximum density equal to 3. . . . .	10
2.5	The REP instance $\mathcal{R}$ constructed from the 3SAT instance $\mathcal{S}$ ( $\mathcal{S}$ contains 3 variables $\{x_1, x_2, x_3\}$ and 3 clauses $\{c_1, c_2, c_3\}$ , where $c_1 = (x_1 \vee \bar{x}_2 \vee x_3)$ , $c_2 = (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$ , and $c_3 = (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$ ). . . . .	14
2.6	The whole rectangular region $R$ is partitioned into $O(n^2)$ small rectangular regions by the cut-lines obtained by extending the boundary intervals of the $n$ rectangles. . . . .	17
2.7	A REP instance containing 10 rectangles. The density $d_{max}$ of its optimal solution is 2. . . . .	21
2.8	The 4-approximation solution of the REP instance in Figure 2.7. . . . .	21
2.9	The rectangle $r_5$ is set to escape to the top boundary since the least area is occupied in this way. . . . .	23
2.10	Bus escape routes in different directions may occupy different numbers of layers. . . . .	27
2.11	Simultaneous REP. . . . .	28
2.12	The bus escape solution generated by REPApx for test case Ex2 with 20 rectangles ( $d_{max} = 2$ ). . . . .	33
2.13	An example REP instance with 9 rectangles. The solution obtained by the greedy approach is shown in (a), while the optimal solution is shown in (b). . . . .	33
3.1	Simultaneous escape routing. . . . .	35
3.2	A routing tile of $cap(2,2,3)$ : (a) horizontal capacity, (b) vertical capacity, (c) diagonal capacity. . . . .	36
3.3	The graph model for one tile. . . . .	37

3.4	Illustration for the graph model's (a) horizontal capacity, (b) vertical capacity and (c) diagonal capacity. . . . .	38
3.5	Proof of Theorem 5: (a) horizontal min-node-cut, (b) vertical min-node-cut, (c) diagonal min-node-cut. . . . .	38
3.6	The graph model for a routing tile of $cap(2,2,4)$ . . . . .	39
3.7	The whole routing graph. . . . .	40
3.8	Bus untangling: (a) single detour untangling vs. (b) two sides untangling. Untangling on both sides reduces the maximum track usage on one side. . . . .	42
3.9	NCER's routing solution of a simultaneous escape routing problem with 45 nets. . . . .	44
3.10	Applying NCER on bus untangling problem: result of NCER (b) has less maximum track usage than that of single detour untangling (a). . . . .	45
3.11	Applying NCER on bus untangling problem: NCER solves problem that cannot be solved by single detour untangling. . .	46
4.1	A sample net-centric escape routing solution for a problem with two buses. Nets of these two buses are mixed up. . . . .	48
4.2	(a) illustrates the bus projection rectangle; (b), (c) and (d) show the bus decomposition issue: the large bus in (b) cannot be routed within its projection rectangle on one layer, so it is decomposed into two smaller buses ((c) and (d)), each of which is able to be routed on one layer. However, the two resultant buses are required to be routed on consecutive layers. . . . .	49
4.3	The two buses can be routed on one layer if bus $b_1$ is escaped to the top boundary. . . . .	50
4.4	(a) Internal conflict, (b) external conflict, (c) a feasible set. . .	52
4.5	Buses without internal conflict have external conflict. . . . .	55
4.6	Clockwise traversal $T_a$ and clockwise traversal $T_b$ . . . . .	57
4.7	The search tree structure. . . . .	58
4.8	Sample of branch-and-bound method. . . . .	58
4.9	Computing LCS when there are pre-assigned buses. . . . .	60
4.10	(a) A bus escape routing instance. (b) The correlation graph constructed. (c) The correlation graph after merging; the vertices representing groups and the ones representing other buses are separated. (d) Applying the min-cut algorithm. (e) Applying the min-cut algorithm again. . . .	65
5.1	Layout of a GNRFET design. (a) Optical image of the device layout. (b) Scanning electron microscopy image of the graphene channel and contacts. (c) Schematic cross section of the graphene transistor. . . . .	68



5.2	Armchair edge states lead to semiconductive GNRs while zigzag edge states lead to metallic GNRs. . . . .	70
5.3	All-graphene GNR-FET design: the metal-semiconductor junction is formed by the change of GNR chirality. . . . .	71
5.4	GNR orientations: (a) semiconducting, (b) metallic. . . . .	72
5.5	Effect of different bends on GNR chirality. . . . .	73
5.6	Triangular routing grid for GNR routing path. The cost of a path depends on the length and bending. . . . .	74
5.7	Modeling the bending cost at a node by expanding the node into six nodes, adding edges between the six nodes and assigning proper costs to the added edges. . . . .	76
5.8	For a terminal, we add a node in the center in addition to the node expansion. Instead of connecting the six nodes, we connect the center node to the six nodes using edges of proper costs. . . . .	77
5.9	The routing on the mesh $M$ (left) and the corresponding path in the expanded graph $G'$ (right). . . . .	78
5.10	Even though the actual route in $M$ (left) has a bending of $120^\circ$ , directly applying the shortest path algorithm will result in a route (dashed route) with bending cost $\alpha_{60^\circ}$ in $G'$ . . . . .	78
5.11	An example on which the modified Dijkstra's algorithm cannot produce the correct result. . . . .	80
5.12	Construction of nodes and edges in the second graph expansion: (a) expand a node; (b) expand a black edge; (c) expand a red edge. . . . .	81
5.13	Expanded graph of Figure 5.7. . . . .	81
5.14	Expanded graph of Figure 5.8. . . . .	82
5.15	Construction of nodes and edges for directed graph: (a) expand a black edge; (b) expand a red edge. . . . .	85
5.16	The resultant routing of $ex3$ . . . . .	87
6.1	(a) The original $s$ - $t$ connection is a path (the path with thick edges), $Pr(s, t) = 81.45\%$ ; (b) Two redundancy segments are added into the $s$ - $t$ connection, $Pr(s, t) = 99.05\%$ ; (c) One more redundancy segment is added, and $Pr(s, t)$ is nontrivial to compute. . . . .	92
6.2	Series composition and parallel composition. . . . .	94
6.3	Illustration of the flow network construction. . . . .	97
6.4	Labeling of the vertices on the original $s$ - $t$ path; Multiple redundancy segments connecting $v_i$ and $v_j$ may exist. . . . .	99
6.5	The optimal solution contains $g_{1,n}$ . . . . .	100
6.6	The optimal solution does not contain $g_{1,n}$ . . . . .	101

6.7	Experimental results on the 3D grid graph with uniform edge costs and survival probabilities. The tradeoff between $Pr(s, t)$ and $B$ is visualized with a curve for each $Pr(e)$ . . . .	106
7.1	(a) The routing solution cannot be successfully decomposed due to the conflict; (b) The routing solution can be decomposed at the expense of introducing a stitch; (c) The routing solution can be decomposed without stitching. . .	109
7.2	Graph model for a non-pin vertex in the routing grid. . . . .	112
7.3	Graph model for a pin vertex in the routing grid. . . . .	112
7.4	(a) Routes on the original routing grid; (b) Routes on the expanded routing graph. . . . .	113
7.5	A path in the expanded routing $G'$ may correspond to a walk with loop in the routing grid $G$ . . . . .	115
7.6	(a) Pattern to be printed; (b) Stitching at corner results in significant printability degradation due to overlay error and line-end effect. . . . .	115
7.7	Corner stitches cannot be prevented by simply removing the edges that directly generate corner stitches. . . . .	116
7.8	Graph model for TPL that disallows stitch at corner. . . . .	117
7.9	Graph model for DPL that disallows stitch at corner. . . . .	118
7.10	An example using the negotiated congestion based scheme to handle conflicts. (a) Nets 1-5 are routed and colored, and net 6 is not routed yet; (b) Net 6 is routed in red, causing conflicts with net 4; (c) In the next iteration, net 4 is rerouted in blue due to the penalty exerted on the red edges. . . . .	123

# LIST OF ABBREVIATIONS

CPU	Central Processing Unit
DPL	Double Patterning Lithography
EDA	Electronic Design Automation
FET	Field-Effect Transistor
GB	Gigabyte
GHz	Gigahertz
GNR	Graphene Nanoribbon
ILP	Integer Linear Programming
LP	Linear Programming
MC	Minimum Coloring
MCM	Multi-Chip Module
MHCP	Minimum Hybrid Cost Path
MIS	Maximum Independent Set
NCER	Negotiated Congestion based Escape Router
NP	Non-deterministic Polynomial
PCB	Printed Circuit Board
REP	Rectangle Escape Problem
TPL	Triple Patterning Lithography

# CHAPTER 1

## INTRODUCTION

In electronic design automation (EDA), routing is one of the most important tasks for both printed circuit boards (PCB) and integrated circuits (IC). After placement, which determines the location of each component on a PCB or element of an IC, the routing step adds wires needed to properly connect the placed components/elements while obeying all the design rules. The quality of routing has a great impact on the performance of the board/chip.

### 1.1 Background and Motivation

A modern PCB usually hosts several chip packages whose footprints on board are arrays of pins. PCB routing is the problem of determining wiring connections between pin terminals on the circuit board. Today's high-end PCBs can have over 2000 pins [1]. A typical high-end PCB could have more than 10 thousand signal nets. On the other hand, the size of a package is kept to a minimum. This makes the footprint of such a package on a PCB a very dense pin grid. Such a large net count and high pin density make manual design of PCBs an extremely time-consuming and error-prone task. In addition, the planar routing constraint makes the problem more difficult. The pins on the board are expected to be connected by non-crossing wires. Not all connections can be routed on one layer, so we may need multiple layers to accommodate all the wire connections. However, introducing vias at the middle of a route would introduce reflection and ringing effects which can cause serious signal integrity issues [2]. Therefore, it is highly preferred that no vias are inserted in the middle of the routing (vias are allowed at the two ends to connect to the package pin). This requires the routing of a net to be planar without switching layers. This planar routing style makes PCB routing a unique problem. For this reason, conventional IC routing algo-

rithms cannot be applied to solve the PCB routing problem. Moreover, we observed from industrial manual solutions that nets are usually grouped as buses on PCBs. Nets belonging to one bus usually have similar timing and other constraints and are thus expected to be routed close to each other on the same layer. Due to the huge pin count and high density of the pin array, it usually requires multiple layers to route the buses without any conflict. In fact, modern PCBs may contain as many as 20 layers [3]. Also, the number of layers needed wants to be minimized as the fabrication cost dramatically increases with the number of layers. Therefore, how to assign the routing of buses to different layers also becomes an important issue. To the best of our knowledge, no mature commercial automated router can handle these problems well. Therefore, further research efforts need to be devoted to the automation of PCB routing, on both bus level and net level.

IC routing, on the other hand, has been extensively studied [4–7]. It is usually divided into global routing and detail routing, which respectively determine the topological routes and exact geometries of the wires. Although IC routing is a theoretically hard problem, it can be handled well by the state-of-the-art commercial routers. However, new routing related problems arise as the feature sizes keep scaling down.

For instance, conventional CMOS devices are facing an increasing number of challenges as the feature sizes scale down. In the meantime, new nanoscale materials, graphene nanoribbons (GNR) in particular, have been shown to have favorable device characteristics and large integration capability, and thus have great potential to revolutionize the fabrication and integration of electronic systems and operate beyond the perceived scaling limitations of traditional CMOS [8]. The graphene nanoribbons are also found to possess metallic properties and thus can be used as interconnection wiring [9]. Due to the unique benzene-like hexagonal ring structure of graphene, metallic zig-zag GNRs can be of three different orientations, implying a routing grid very different from the rectangular grid for conventional metal routing. Obviously, the maze routing algorithm pervasively used in traditional IC routers is not applicable to such a routing problem. We need a new routing model to capture the unique routing grid and delay metric. In addition, the GNR wire segments have a connection defective rate. Some GNR wire segments will be more vulnerable to wear-out effects, and some wire segments may fail to connect during manufacturing [9]. In such a circumstance, an intercon-

nection between two terminals realized by a simple path, like in traditional IC routing, is vulnerable to defect, since this interconnection fails as long as any wire segment in this path fails. This definitely creates more challenges for routing reliability, and inspires us to develop new routing strategies.

Another IC routing related problem emerges due to the manufacturing issues. As the minimum feature size keeps shrinking, and the availability of the next generation lithography methods (EUV, e-beam direct write, etc.) further delays, double patterning lithography (DPL) is commonly recognized as a feasible lithography process for 20nm technology nodes [10–13]. As technology continues to scale, double patterning lithography (DPL) is pushed to near its limit. triple patterning lithography (TPL) is a considerable and natural extension along the paradigm of DPL. With an extra mask to accommodate the features, TPL can be used to eliminate the unresolvable conflicts and minimize the number of stitches, which are pervasive in the DPL process, and thus greatly improve the layout decomposability. It is observed that most hard-to-decompose features are generated during routing, so considering TPL during the routing stage can open a much larger solution space and thus can further benefit the layout decomposition step. As far as we know, no existing works address the problem of TPL aware routing.

## 1.2 Overview of this Dissertation

In this dissertation, we first present our research results on PCB routing.

The works on bus-based PCB routing are presented in Chapter 2 and Chapter 4. In PCB routing, we want to minimize the number of layers as the fabrication cost dramatically increases with the number of layers. In Chapter 2, we introduce and study the Rectangle Escape Problem (REP), which is motivated by bus-based escape routing within one component on a PCB. This problem is basically to determine an escape direction for each bus within the component such that the resultant maximum density (a good indicator of the number of layers needed) is minimized. We prove that the REP is NP-complete, and show that it can be formulated as an Integer Linear Program (ILP). A provably good approximation algorithm for the REP is developed by applying Linear Programming (LP) relaxation and a special rounding technique to the ILP. In addition, an iterative refinement procedure

is proposed as a postprocessing step to further improve the results. Our approximation algorithm is also shown to work for more general versions of REP: weighted REP and simultaneous REP. Our approach is tested on a set of industrial PCB bus escape routing problems. Experimental results show that the optimal solution can be obtained within several seconds for each of the test cases. In Chapter 4, we further study the optimal layer assignment of a set of buses connecting two components on PCB. This is a theoretically hard problem and we propose a branch-and-bound based algorithm that optimally solves it. Our algorithm is guaranteed to produce a feasible layer assignment of the buses with a minimum number of layers. We applied our algorithm on industrial data and the experimental results validate our approach.

Chapter 3 focuses on net-based PCB routing. The negotiated congestion based routing scheme finds success in FPGA routing and IC global routing. However, its application in simultaneous escape routing, a key problem in PCB design, has never been reported in the literature. In this chapter, we investigate how well the negotiated congestion based router performs on escape routing problems. We propose an underlying routing graph which correctly models the routing resources of the pin grids on board. We then build a Negotiated Congestion based Escape Router (NCER) by applying the negotiated congestion routing scheme on the constructed routing graph. We compare the performance of NCER with that of the Cadence PCB router Allegro on 14 industrial test cases, and experimental results show that the two routers have comparable routability: each completely routes 7 test cases. Moreover, we observe that NCER and Allegro exhibit complementary behaviors: each is able to solve most of the test cases that the other cannot solve. Together, they completely route 11 test cases. Therefore, by using NCER as a supplement to Allegro, we can solve a broader range of escape routing problems.

We then present our results on the emerging problems in IC routing.

Conventional CMOS devices face an increasing number of challenges as their feature sizes scale down. Graphene nanoribbon (GNR) based devices are shown to be a promising replacement of traditional CMOS at future technology nodes. However, all previous works on GNRs focus at the device level. In order to integrate these devices into electronic systems, routing becomes a key issue. In Chapter 5, the GNR routing problem is studied for the first time. We formulate the GNR routing problem as a minimum

hybrid-cost shortest path problem on triangular mesh (“hybrid” means that we need to consider both the length and the bending of the routing path). We show that by graph expansion, this minimum hybrid-cost shortest path problem can be solved by applying the conventional shortest path algorithm on the expanded graph. Experimental results show that our GNR routing algorithm effectively handles the hybrid cost. We then study the routing reliability problem in Chapter 6. In practice, the GNR wire segments can have a connection defective rate. Particularly, each wire segment has a survival probability, and thus has a chance to fail, making traditional routing very unreliable. In Chapter 6, we introduce the routing reliability problem and propose an algorithm flow to solve it. Given an  $s$ - $t$  routing path on a routing graph, we try to reinforce the reliability of the routing path by adding redundant wiring segments in such a way that its survival probability is maximized with a reasonable overhead of routing resources. Our proposed algorithm flow is two-fold: (1) generation of candidate redundancy segment via min-cost max-flow; (2) optimal selection among the candidates by dynamic programming. The results of extensive experiments confirm the effectiveness and efficiency of our approach.

In Chapter 7, we study the problem of triple patterning lithography (TPL) aware routing. We first propose a graph model that correctly models the cost of conflicts and stitches in TPL. By replacing each vertex in the routing grid with the graph model and performing shortest path algorithm on the expanded graph, the optimal path with mask/color assignment for one net can be computed, in the presence of previously routed and colored nets. Our proposed graph model is a unified model that can be extended to handle multiple patterning and can also be tailored for double patterning. We then develop a negotiated congestion based scheme to resolve conflicts. The regions with conflicts are penalized and the nets are iteratively rerouted and re-colored. Experimental results show that this scheme is very effective and all conflicts can be resolved within a small number of iterations in our test cases. An effective heuristic is also developed to ensure approximately balanced utilization of the three masks. To the best of our knowledge, this is the first triple patterning aware detailed router developed in the literature. We also implement a double patterning aware detailed router by adapting our graph model for double patterning, and compare it with the triple patterning version in 14nm node. Experimental results show that, using TPL, the con-



flicts can be resolved much more easily and the stitches can be significantly reduced in contrast to DPL.

## CHAPTER 2

# BUS-BASED ESCAPE ROUTING ON PCB

### 2.1 Introduction

In the past few years, as the dimensions of packages and PCBs keep decreasing and the pin counts and routing layers keep increasing, the escape routing problem, which is to route nets from their pins to the component boundaries, becomes more and more critical [14–16]. In a PCB bus escape routing instance, the nets of a bus are preferred to be routed together, without mixing with the nets from other buses [17–19]. From industrial manual routing solutions, we observe that the escape routes of all the nets of a bus are typically within one of its projection rectangles, which can be obtained by extending the bounding box of the pin cluster of the bus to one of the component boundaries. A projection rectangle of a bus is defined as the rectangular region including the bounding box of the pin cluster of the bus as well as the extension part obtained by projecting the bounding box to one of the boundaries. The shaded region in Figure 2.1 demonstrates a projection rectangle of a bus. Figure 2.2 shows the four projection rectangles of a bus, together with an example escape routing to the right boundary. The bound-

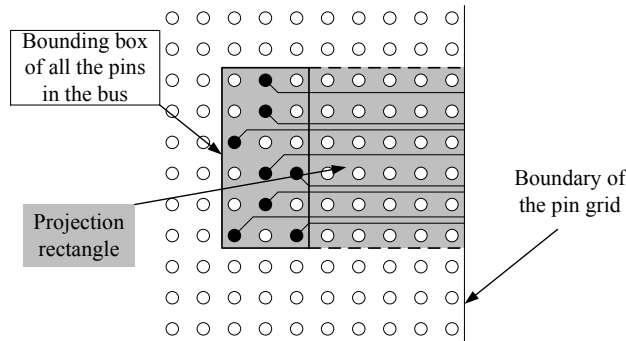


Figure 2.1: The projection rectangle is obtained by projecting the bounding box of the pin cluster of the bus to one of the boundaries of the component.

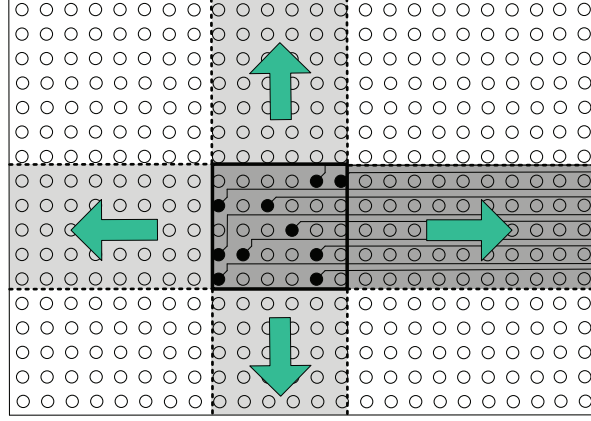


Figure 2.2: The four projection rectangles of a bus and its escape routing to the right boundary of the component.

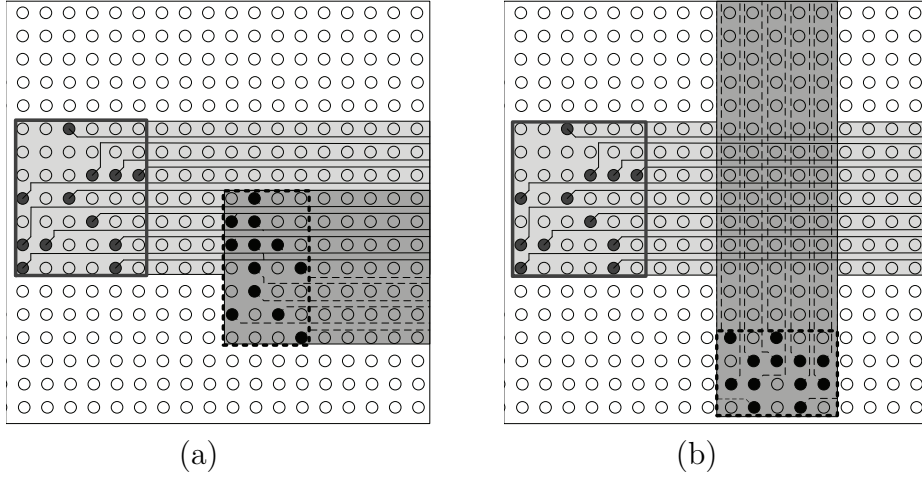


Figure 2.3: The routes of two buses conflict.

ing box of the pin cluster of a bus can be represented by a rectangle in REP, while the four projection rectangles correspond to the escapes in four directions. In [20], Kong *et al.* compare their bus projection based escape routing solution with the net based escape routing solution and observe that the bus projection based solution usually results in shorter wire length and occupies smaller routing space. Moreover, the wire length of the nets within a bus are usually required to be approximately identical [21], due to the timing issue. The bus projection based routing style produces much more balanced wire length, making length-matching routing an easier problem [21].

When the escape routes of two buses conflict, they have to be routed on different layers. Figure 2.3 demonstrates two types of conflict. The fabri-

cation cost dramatically increases when more layers are needed, so we want to use as few layers as possible to accommodate all the buses. According to our experience, the maximum density is usually a good indicator of the number of layers needed (although theoretically it is only a lower bound of the necessary number of layers). Therefore, in our Rectangle Escape Problem, we try to minimize the maximum density. Note that a whole PCB board contains a number of components (pin grids), and by solving the Rectangle Escape Problem for each component, we can obtain a bus escape planning for the whole PCB board. After that, the corresponding buses of different components can further be connected and the actually layer assignment can be performed.

In this chapter, we model this bus-based escape routing problem as Rectangle Escape Problem (REP). In REP, we are given a rectangular region  $R$  and a set  $S$  of rectangles staying within  $R$ , where each rectangle has to “escape” to one of the four boundaries of  $R$ . By “escape” we mean projecting the rectangle onto one of the four boundaries of  $R$ , namely, left, right, top or bottom. The resultant rectangle (including the region of the original rectangle) is called a projection rectangle. The objective of REP is to determine an escape direction for each rectangle in  $S$ , such that the resultant maximum density over the region  $R$  is minimized. In the region  $R$ , the density at a point is the number of projection rectangles containing this point, and the point with largest density defines the maximum density over  $R$ . Figure 2.4 illustrates the Rectangle Escape Problem: Figure 2.4 (a) shows the input rectangles in  $S$ , while Figure 2.4 (b) and (c) give two escape solutions to this problem. The shaded region attached to each rectangle is the extension of the corresponding rectangle after escaping. It is easy to see that the resultant maximum densities of the two escape solutions in Figure 2.4 (b) and (c) are respectively 2 and 3. The objective of the REP is to minimize the maximum density over the rectangular region  $R$ , so the solution in Figure 2.4 (b) is better than the solution in Figure 2.4 (c).

Our contributions can be summarized as follows:

- We prove that the REP is NP-complete.
- A 4-approximation algorithm for REP is developed.
- This approximation algorithm is also shown to work for more general

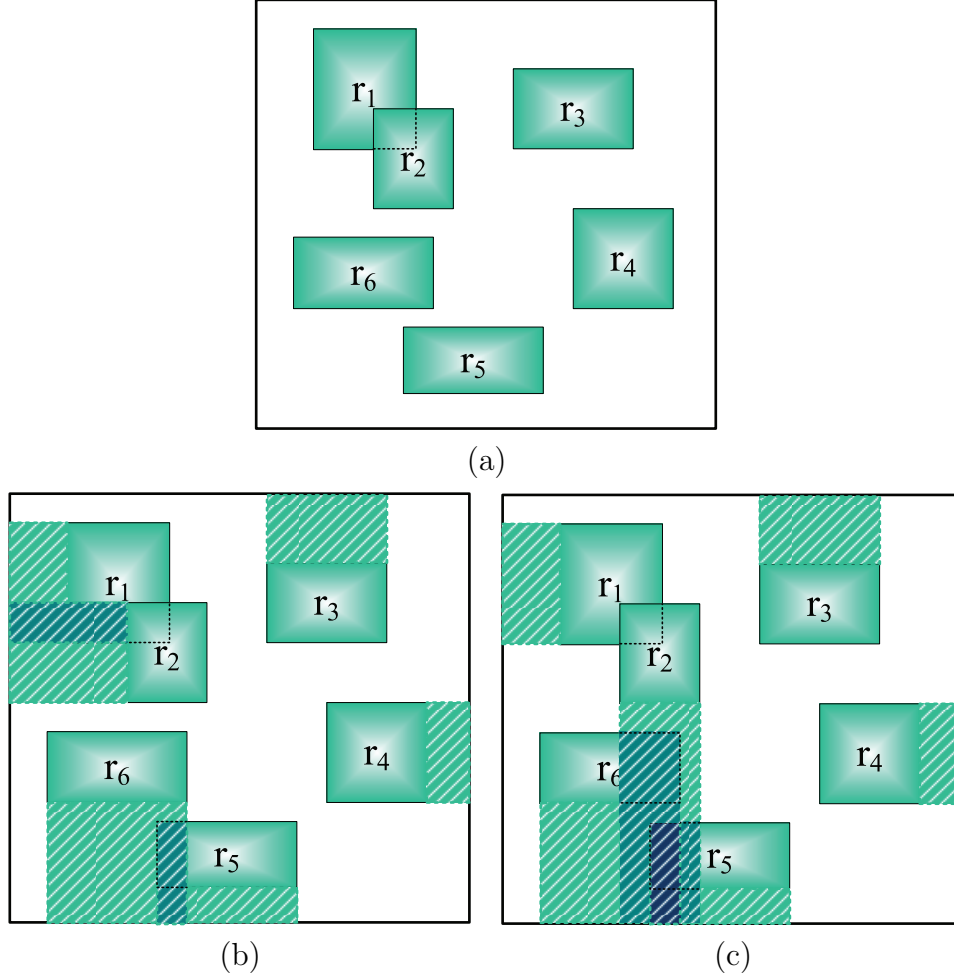


Figure 2.4: Illustration of Rectangle Escape Problem: (a) input rectangles; (b) an escape solution with maximum density equal to 2; (c) an escape solution with maximum density equal to 3.

versions of REP: weighted REP and simultaneous REP.

- Our algorithm is tested on a set of industrial PCB bus escape routing cases, and results show that the optimal solution can be obtained within several seconds for each test case, which confirms the effectiveness and efficiency of our approach.

The remainder of this chapter is organized as follows. The REP is defined in Section 2.2. We will prove the NP-completeness of the REP in Section 2.3, and present our approximation algorithm for the REP in Section 2.4. Experimental results are reported in Section 2.6 before concluding in the last section. This work is published in [22] and [23].

## 2.2 Problem Definition

An REP instance  $\mathcal{R}$  contains a rectangular region  $R$  and a set  $S$  of  $n$  rectangles  $\{r_1, r_2, \dots, r_n\}$ . Given a candidate escape solution for the instance  $\mathcal{R}$ , let  $d_{max}$  denote the resultant maximum density over the rectangular region  $R$ , which is defined as follows:

**Definition 1. The maximum density  $d_{max}$**  - *In the rectangular region  $R$ , the density at a point is the number of projection rectangles containing this point, and the point with largest density defines the maximum density  $d_{max}$  over  $R$ .*

We can now define the Rectangle Escape Problem.

### **RECTANGLE ESCAPE PROBLEM (REP)**

INSTANCE: A rectangular region  $R$  and a set  $S$  of  $n$  rectangles  $\{r_1, r_2, \dots, r_n\}$  residing within  $R$ .

QUESTION: Each rectangle  $r_i \in S$  chooses a direction to escape, such that  $d_{max}$  is minimized.

## 2.3 Proof of NP-Completeness

In this section, we show that REP is NP-complete. In order to facilitate the proof of NP-completeness, the decision version of REP is described as follows:

### **DECISION VERSION OF REP**

INSTANCE: An integer  $k$ , a rectangular region  $R$  and a set  $S$  of  $n$  rectangles  $\{r_1, r_2, \dots, r_n\}$  residing within  $R$ .

QUESTION: Each rectangle  $r_i \in S$  chooses a direction to escape, is there a solution such that  $d_{max} \leq k$ ?

**Lemma 1.** *REP is in NP.*

*Proof.* Given an escape solution for the REP, where each rectangle  $r_i$  has its escape direction determined, we only need to show that the resultant

maximum density  $d_{max}$  can be checked in polynomial time. Note that  $d_{max}$  equals the maximum clique size of the rectangle intersection graph of the  $n$  rectangles (after escaping). Lee showed in [24] that the maximum clique of a rectangle intersection graph can be computed in  $O(n \log n)$  time.  $\square$

We then prove REP is NP-Hard by using reduction from *3SAT*, which is a classical NP-complete problem [25].

**3SAT**

INSTANCE: A set  $U$  of  $n$  variables  $\{x_1, x_2, \dots, x_n\}$ , a collection  $C$  of  $m$  clauses  $\{c_1, c_2, \dots, c_m\}$  over  $U$  such that  $|c_i| = 3$ , for  $1 \leq i \leq m$ .

QUESTION: Is there a satisfying assignment for  $C$ ?

Given a *3SAT* instance  $\mathcal{S}$ , we construct an REP instance  $\mathcal{R}$ , such that  $\mathcal{S}$  has a satisfying truth assignment if and only if  $\mathcal{R}$  has an escape solution with maximum density  $d_{max} \leq 3$ . The constructed REP instance  $\mathcal{R}$  contains three types of rectangles, namely, blockage rectangles, variable rectangles and clause rectangles. The reduction is conducted as follows:

1. A rectangular region  $R$  is created at first. Three overlapping blockage rectangles are placed along the top boundary of  $R$  and another three overlapping blockage rectangles are placed along the left boundary of  $R$ , so that the top boundary and the left boundary are “blocked”. The bottom boundary and the right boundary of  $R$  are referred to as the “True” boundary and the “False” boundary, respectively (see Figure 2.5).
2. The variable rectangles are placed along the diagonal of  $R$ . For each variable  $x_i$ , we create a rectangle  $x_i$  and a rectangle  $\bar{x}_i$ , and place them in such a way that the lower right corner of rectangle  $x_i$  overlaps with the upper left corner of rectangle  $\bar{x}_i$ . Note that the projection of variable rectangles for  $x_i$  and the projection of variable rectangles for  $x_j$  cannot overlap with each other (on either the “True” boundary or the “False” boundary), if  $i \neq j$ . In addition, for each variable  $x_i$ , we place two overlapping blockage rectangles along the “True” boundary, with their horizontal position as the projection of the two variable rectangles  $x_i$  and  $\bar{x}_i$  on the “True” boundary; similarly, we place another two overlapping blockage rectangles along the “False” boundary, with their

vertical position as the projection of the two variable rectangles  $x_i$  and  $\bar{x}_i$  on the “False” boundary (see Figure 2.5 for illustration).

3. The clause rectangles are placed below all the variable rectangles. For each clause  $c_i$ , we create three copies of clause rectangle for  $c_i$ . If clause  $c_i$  contains  $x_j$ , we place one copy of clause rectangle  $c_i$  below the variable rectangle  $\bar{x}_j$ ; if clause  $c_i$  contains  $\bar{x}_j$ , we place one copy of clause rectangle  $c_i$  below the variable rectangle  $x_j$ . Note that the clause rectangle  $c_i$  cannot be placed below the overlapping region of the two variable rectangles  $x_j$  and  $\bar{x}_j$ , and that the three copies of clause rectangle for  $c_i$  should be placed with the same vertical coordinate. In addition, a blockage rectangle for  $c_i$  needs to be placed along the “False” boundary, with its vertical position as the projection of the clause rectangles for  $c_i$  on the “False” boundary (see Figure 2.5 for illustration). Furthermore, we should make sure that the projection of a clause rectangle for  $c_i$  and the projection of a clause rectangle for  $c_j$  do not overlap (on either the “True” boundary or the “False” boundary), if  $i \neq j$ .

It is easy to see that the reduction can be done in polynomial time with respect to the size of the 3SAT instance  $\mathcal{S}$ , so we have the following Lemma.

**Lemma 2.** *The reduction from a general 3SAT instance  $\mathcal{S}$  to a REP instance  $\mathcal{R}$  can be done in polynomial time with respect to the size of the 3SAT instance  $\mathcal{S}$ .*

Figure 2.5 shows the REP instance  $\mathcal{R}$  constructed from a concrete 3SAT instance  $\mathcal{S}$  with 3 variables  $\{x_1, x_2, x_3\}$  and 3 clauses  $\{c_1, c_2, c_3\}$ , where  $c_1 = (x_1 \vee \bar{x}_2 \vee x_3)$ ,  $c_2 = (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$ , and  $c_3 = (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$ .

**Lemma 3.** *The 3SAT instance  $\mathcal{S}$  has a satisfying truth assignment if and only if the constructed REP instance  $\mathcal{R}$  has an escaping solution with  $d_{max} \leq 3$ .*

*Proof.* (if part) If the constructed REP instance  $\mathcal{R}$  has an escaping solution with  $d_{max} \leq 3$ , we can obtain a satisfying truth assignment for the 3SAT instance  $\mathcal{S}$ . For each variable  $x_i$ , either the variable rectangle  $x_i$  escapes to the “True” boundary (the variable rectangle  $\bar{x}_i$  escapes to the “False” boundary), or the variable rectangle  $x_i$  escapes to the “False” boundary (the



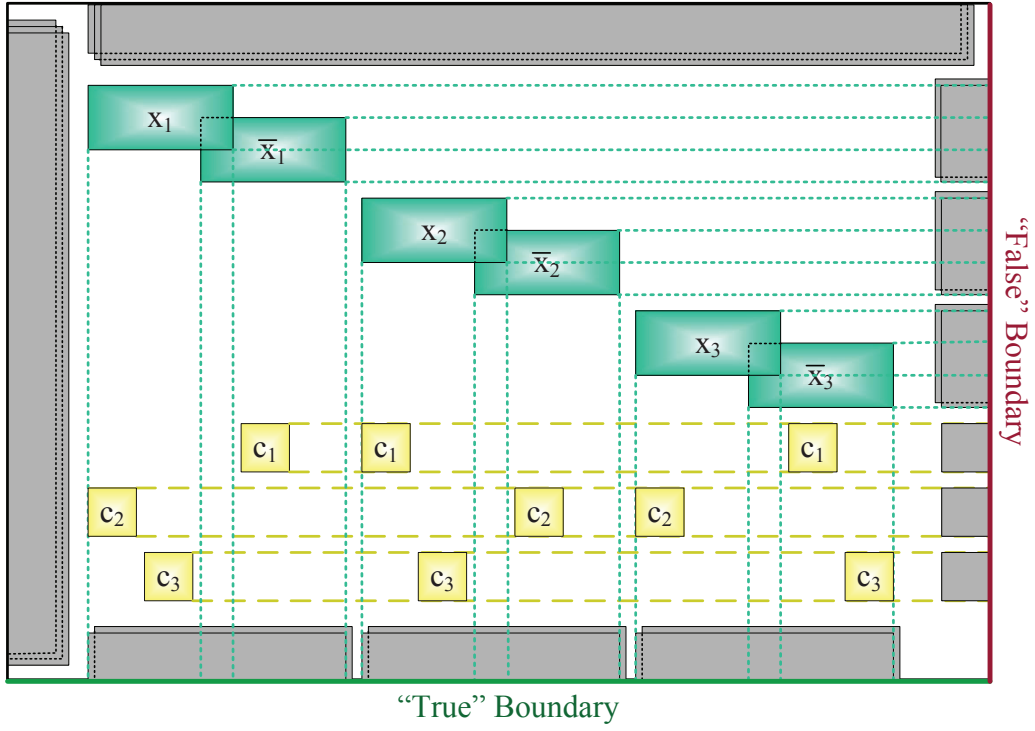


Figure 2.5: The REP instance  $\mathcal{R}$  constructed from the 3SAT instance  $\mathcal{S}$  ( $\mathcal{S}$  contains 3 variables  $\{x_1, x_2, x_3\}$  and 3 clauses  $\{c_1, c_2, c_3\}$ , where  $c_1 = (x_1 \vee \bar{x}_2 \vee x_3)$ ,  $c_2 = (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$ , and  $c_3 = (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$ ).

variable rectangle  $\bar{x}_i$  escapes to the “True” boundary), since if they escape to the same boundary,  $d_{max}$  is at least 4, due to the blockage rectangles. We set  $x_i$  to be true if the variable rectangle  $x_i$  escapes to the “True” boundary, and set it to be false otherwise. We claim this is a satisfying truth assignment of  $\mathcal{S}$ . For the sake of contradiction, suppose there is a clause  $c_i$  which is not satisfied by this assignment. Without loss of generality, let us assume that clause  $c_i$  contains variable  $x_j$ . According to our construction of the REP instance, one of the three copies of clause rectangle  $c_i$  is placed below the variable rectangle  $\bar{x}_j$ . Since clause  $c_i$  is not satisfied,  $x_j$  must be set to false, which indicates that the variable rectangle  $x_j$  escapes to the “False” boundary and the variable rectangle  $\bar{x}_j$  escapes to the “True” boundary. Therefore, this copy of clause rectangle  $c_i$  can only escape to the “False” boundary, as escaping to the “True” boundary makes the local density become 4 (2 blockage rectangles along the “True” boundary, the variable rectangle  $\bar{x}_j$  and this copy of clause rectangle  $c_i$ ). Based on the same argument, the other two copies of clause rectangle  $c_i$  have to escape to the “False” boundary, too. However, if all the three copies of clause rectangle  $c_i$  escape to the “False” boundary, it also makes the local density become 4 (one blockage rectangle along the “False” boundary, and the three copies of clause rectangle  $c_i$ ). Now, we can conclude that  $d_{max}$  will be at least 4 no matter how we escape the three copies of clause rectangles  $c_1$ , which contradicts to the existence of an escaping solution with  $d_{max} \leq 3$ .

(only if part) If the 3SAT instance  $\mathcal{S}$  has a satisfying truth assignment, we can obtain an escaping solution with  $d_{max} \leq 3$  for the constructed REP instance. For each variable  $x_i$ , we let the two corresponding variable rectangles  $x_i$  and  $\bar{x}_i$  escape to the “True” (“False”) boundary and the “False” (“True”) boundary, respectively, if variable  $x_i$  is set to be true (false). For each copy of clause rectangle  $c_i$ , we let it escape to the “True” boundary if this does not make  $d_{max}$  exceed 3, otherwise we let it escape to the “False” boundary. Now we have an escape solution for  $\mathcal{R}$ , and it is easy to see  $d_{max} \leq 3$  for this escape solution.  $\square$

Combining the three Lemmas, we have proved that REP is NP-Complete.

**Theorem 1.** *REP is NP-complete.*

## 2.4 The Algorithm

In this section, we first show that the REP can be formulated into an integer linear programming (ILP); then we demonstrate that a 4-approximation algorithm can be obtained by using linear programming (LP) relaxation and rounding technique. Furthermore, we introduce a more general version of REP with weights, namely, weighted REP, and show that the 4-approximation algorithm also works for weighted REP. Finally, an iterative refinement procedure is proposed as a postprocessing step to further improve the results.

### 2.4.1 ILP Formulation

We first introduce some notations in order to facilitate the ILP formulation. Given an REP instance  $\mathcal{R}$ :

- We introduce four 0-1 variables for each rectangle  $r_i \in S$ , namely,  $x_{il}, x_{ir}, x_{it}$  and  $x_{ib}$ , and we call them direction variables. The variable  $x_{il}(x_{ir}, x_{it}, x_{ib})$  set to be 1 indicates that rectangle  $r_i$  escapes to the Left (Right, Top, Bottom) boundary of the rectangular region  $R$ .
- We extend the four boundary intervals of each rectangle  $r_i \in S$  to the boundaries of the whole rectangular region  $R$ , so that a set of  $O(n)$  horizontal and vertical cut-lines is obtained. Consequently, the rectangular region  $R$  is partitioned into a set  $P$  of  $O(n^2)$  tiles by these cut-lines. Figure 2.6 shows an example consisting of four rectangles.
- For each rectangle  $r_i \in S$ , when it escapes to the boundary of  $R$ , it occupies a larger rectangular region, which is composed of the original region of  $r_i$  and the extension region after escaping. Let  $r_{il}(r_{ir}, r_{it}, r_{ib})$  denote the rectangular region that  $r_i$  occupies after escaping to the Left (Right, Top, Bottom) boundary of  $R$ . Figure 2.6 illustrates the region  $r_{4r}$  and the region  $r_{4b}$  that rectangle  $r_4$  occupies after escaping to the right boundary and the bottom boundary, respectively.

Now we can add our constraints:

- **Direction Constraints**

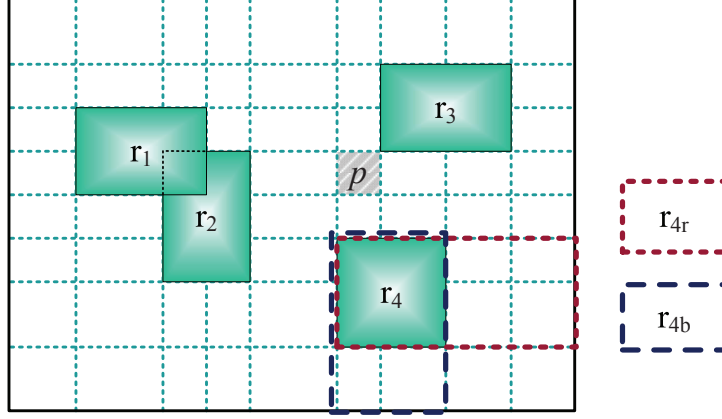


Figure 2.6: The whole rectangular region  $R$  is partitioned into  $O(n^2)$  small rectangular regions by the cut-lines obtained by extending the boundary intervals of the  $n$  rectangles.

Each rectangle  $r_i \in S$  can only choose one direction to escape, so we have the following set of constraints:

$$x_{il} + x_{ir} + x_{it} + x_{ib} = 1, \quad \forall i = 1, 2, \dots, n.$$

- **Density Constraints**

The density within each tile  $p \in P$  should not exceed  $d_{max}$ , so we have the following set of constraints:

$$\sum_{i,* : r_{i*} \text{ occupies } p} x_{i*} \leq d_{max}, \quad \forall p \in P,$$

where ‘\*’ should be replaced by ‘ $l$ ’ (Left), ‘ $r$ ’ (Right), ‘ $t$ ’ (Top), or ‘ $b$ ’ (Bottom), whichever is appropriate.

Let us take the tile  $p$  in Figure 2.6 as an example. It is easy to see that  $r_{1r}$ ,  $r_{2r}$  and  $r_{4t}$  occupy tile  $p$ , so the density constraint for tile  $p$  is added as:

$$x_{1r} + x_{2r} + x_{4t} \leq d_{max}.$$

The objective is to minimize the density  $d_{max}$ , so now the ILP for the REP can be formulated as follows, with  $O(n)$  variables and  $O(n^2)$  constraints:

$$\begin{aligned}
& \text{Minimize} && d_{max} \\
& \text{Subject to} && \\
& && x_{il} + x_{ir} + x_{it} + x_{ib} = 1, \quad \forall i = 1, 2, \dots, n. \\
& && \sum_{i,* : r_{i*} \text{ occupies } p} x_{i*} \leq d_{max}, \quad \forall p \in P. \\
& && x_{il}, x_{ir}, x_{it}, x_{ib} \in \{0, 1\}, \quad \forall i = 1, 2, \dots, n.
\end{aligned}$$

The optimal solution can be obtained by solving the above ILP.

## 2.4.2 A 4-Approximation Algorithm

Solving ILP is also an NP-complete problem [25], and it can be very time-consuming when the problem is large. In the following, we show that a 4-approximation algorithm can be obtained by using LP relaxation and a special rounding technique.

We relax the ILP in Section 2.4.1 into an LP as shown below, which can be efficiently solved by existing LP solvers.

$$\begin{aligned}
& \text{Minimize} && d_{max} \\
& \text{Subject to} && \\
& && x_{il} + x_{ir} + x_{it} + x_{ib} = 1, \quad \forall i = 1, 2, \dots, n. \\
& && \sum_{i,* : r_{i*} \text{ occupies } p} x_{i*} \leq d_{max}, \quad \forall p \in P. \\
& && 0 \leq x_{il}, x_{ir}, x_{it}, x_{ib} \leq 1, \quad \forall i = 1, 2, \dots, n.
\end{aligned}$$

We solve the LP by an LP solver, after which we get a fractional solution (if we are not lucky enough). For each rectangle  $r_i \in S$ , if  $x_{il}$  ( $x_{ir}, x_{it}, x_{ib}$ ) has the largest value among the four direction variables of  $r_i$ , we say  $r_i$ 's dominating escape direction is Left (Right, Top, Bottom), and we call  $x_{il}$  ( $x_{ir}, x_{it}, x_{ib}$ ) the dominating variable of  $r_i$ . For example, if the LP solver produces a solution where  $x_{il} = 0.1$ ,  $x_{ir} = 0.2$ ,  $x_{it} = 0.5$  and  $x_{ib} = 0.2$  for rectangle  $r_i$ , the dominating escape direction of  $r_i$  is Top, and the dominating variable is  $x_{it}$ . We arbitrarily break the tie when two or more direction variables for  $r_i$  have the largest value. Our rounding technique works by,

$\forall r_i \in S$ , simply assigning 1 to the dominating variable, and assigning 0 to the other direction variables, i.e., each rectangle  $r_i$ 's escape direction is set to be its dominating escape direction. This rounding technique gives us a 4-approximation algorithm, LPAPX, as described below.

ALGORITHM LPAPX(REP instance  $\mathcal{R}$ ):

1. Formulate  $\mathcal{R}$  into an ILP.
2. Relax the ILP into an LP.
3. Solve the LP.
4.  $\forall r_i \in S$ , assign  $r_i$ 's escape direction to be its dominating escape direction.
5. Compute the resultant maximum density  $d_{max}$ .

**Theorem 2.** *Given an REP instance  $\mathcal{R}$ , where each rectangle has 4 candidate choices of escape directions, LPAPX is a 4-approximation algorithm for  $\mathcal{R}$ .*

*Proof.* Let  $LP(d_{max})$  and  $APX(d_{max})$  denote the maximum density  $d_{max}$  obtained by the LP (before rounding) and the algorithm LPAPX, respectively, and let  $OPT(d_{max})$  denote the optimal maximum density. Obviously,  $LP(d_{max}) \leq OPT(d_{max})$ , since the solution space increases after LP relaxation.

Now we show that  $APX(d_{max}) \leq 4LP(d_{max})$ .

In step 3 of the algorithm LPAPX, we solve the LP and obtain a fractional solution,  $\{(x_{il}, x_{ir}, x_{it}, x_{ib}), \forall i = 1, 2, \dots, n\}$ . According to the density constraints of the LP,

$$\sum_{i,* : r_{i*} \text{ occupies } p} x_{i*} \leq LP(d_{max}), \quad \forall p \in P. \quad \langle 1 \rangle$$

Let  $\{(\hat{x}_{il}, \hat{x}_{ir}, \hat{x}_{it}, \hat{x}_{ib}), \forall i = 1, 2, \dots, n\}$  denote the integral solution obtained after rounding, so we have

$$\sum_{i,* : r_{i*} \text{ occupies } p} \hat{x}_{i*} \leq APX(d_{max}), \quad \forall p \in P. \quad \langle 2 \rangle$$

There must be a tile  $\hat{p} \in P$  such that the following equality holds:

$$\sum_{i,* : r_{i*} \text{ occupies } \hat{p}} \hat{x}_{i*} = APX(d_{max}). \quad \langle 3 \rangle$$

Let  $D$  denote the set of all the dominating variables. Now let us consider each term  $\hat{x}_{i*}$  on the left-hand side of equation  $\langle 3 \rangle$ . There are two cases:

- 1).  $x_{i*}$  is a dominating variable ( $x_{i*} \in D$ ). In this case,  $x_{i*} \geq 0.25$ . This is because  $x_{il} + x_{ir} + x_{it} + x_{ib} = 1$ ,  $x_{i*}$  must be at least 0.25 to be the dominating variable. Thus,  $\hat{x}_{i*} = 1 \leq 4x_{i*}$ .
- 2).  $x_{i*}$  is not a dominating variable ( $x_{i*} \notin D$ ). In this case,  $\hat{x}_{i*} = 0$ .

Thus, we rewrite equation  $\langle 3 \rangle$  as follows:

$$\begin{aligned}
APX(d_{max}) &= \sum_{i,* : r_{i*} \text{ occupies } \hat{p}} \hat{x}_{i*} \\
&= \sum_{i,* : r_{i*} \text{ occupies } \hat{p}, x_{i*} \in D} \hat{x}_{i*} + \sum_{i,* : r_{i*} \text{ occupies } \hat{p}, x_{i*} \notin D} \hat{x}_{i*} \\
&= \sum_{i,* : r_{i*} \text{ occupies } \hat{p}, x_{i*} \in D} \hat{x}_{i*} + 0 \\
&\leq \sum_{i,* : r_{i*} \text{ occupies } \hat{p}, x_{i*} \in D} 4x_{i*} \\
&\leq 4 \times \sum_{i,* : r_{i*} \text{ occupies } \hat{p}} x_{i*} \\
&\leq 4LP(d_{max}) \\
&\leq 4OPT(d_{max})
\end{aligned}$$

Therefore, LPAPX is a 4-approximation algorithm.  $\square$

Based on the analysis in the proof of Theorem 2, it is easy to see that the approximation ratio of the algorithm LPAPX depends on the number of choices of the escape directions. For example, if each rectangle is only allowed to escape in two directions, LPAPX is a 2-approximation algorithm for REP. Therefore, we have the following immediate corollary:

**Corollary 1.** *Given an REP instance  $\mathcal{R}$ , where each rectangle has  $\alpha$  candidate choices of escape direction, LPAPX is an  $\alpha$ -approximation algorithm for  $\mathcal{R}$ .*

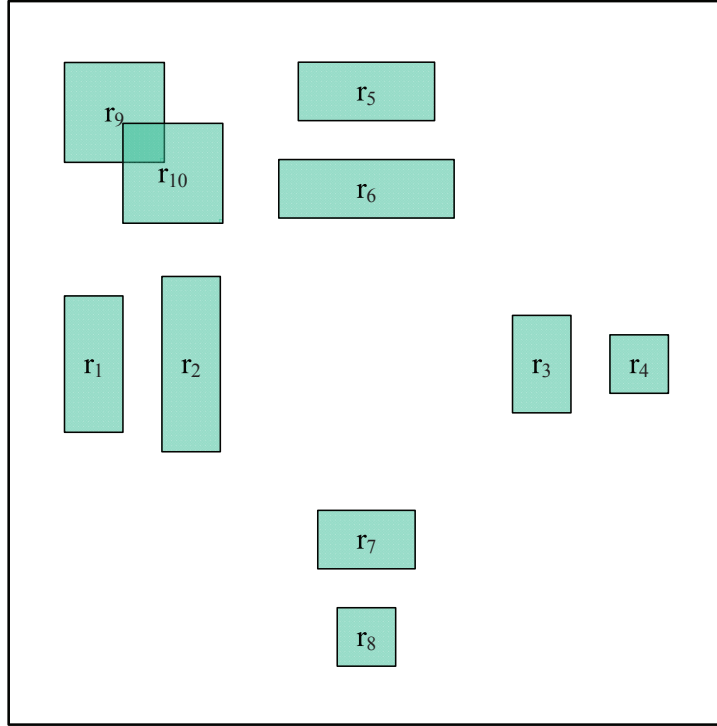


Figure 2.7: A REP instance containing 10 rectangles. The density  $d_{max}$  of its optimal solution is 2.

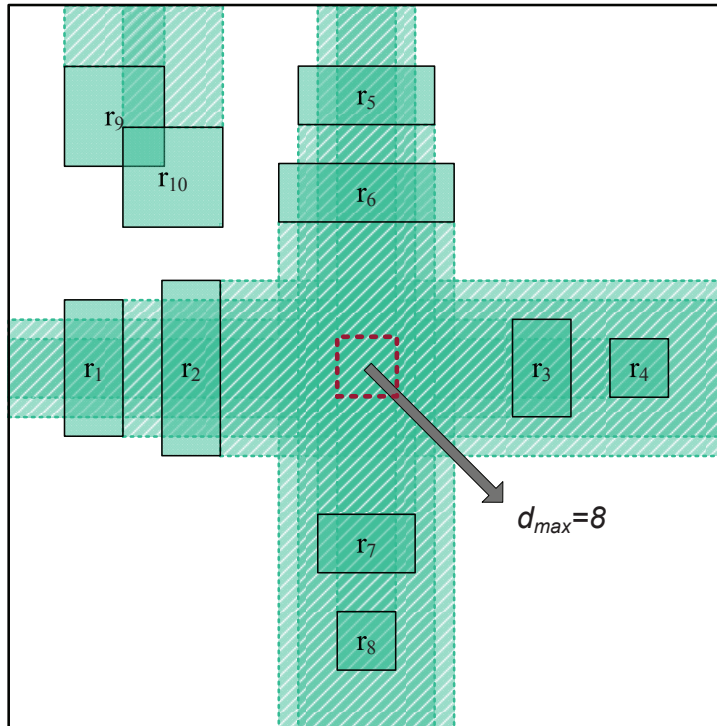


Figure 2.8: The 4-approximation solution of the REP instance in Figure 2.7.



### 2.4.3 Tightness of the Approximation Ratio

In the following, we show that the approximation ratio 4 is tight by constructing a REP instance where the density  $d_{max}$  of the solution obtained by LPAPX is exactly 4 times that of the optimal solution. Figure 2.7 shows a REP instance containing 10 rectangles. It is easy to see that the density  $d_{max}$  of the optimal solution of this instance is 2, which is determined by the overlapping of  $r_9$  and  $r_{10}$ . We show in the following that we can obtain a solution with  $d_{max} = 8$  using LPAPX. We use an LP solver to solve the relaxed ILP of this REP instance. Notice that the set of optimal solutions of this relaxed ILP includes a fractional solution where  $x_{il} = x_{ir} = x_{it} = x_{ib} = 0.25$ ,  $\forall i = 1, 2, \dots, 8$ ,  $r_{it} = 1$ ,  $\forall i = 9, 10$ . This corresponds to an escape solution where  $r_i$  escapes a “quarter” to each of the four directions,  $\forall i = 1, 2, \dots, 8$ ,  $r_9$  and  $r_{10}$  escape to the top. It is possible that the LP solver returns the above-mentioned fractional solution. LPAPX then decides an escape direction for each rectangle using the rounding technique. As there is no dominating escape direction for  $r_1, r_2, \dots, r_8$ , LPAPX will arbitrarily pick an escape direction for them. In the worst case, the escape directions of  $r_1$  and  $r_2$  ( $r_3$  and  $r_4$ ,  $r_5$  and  $r_6$ ,  $r_7$  and  $r_8$ ) are set to be Right (Left, Bottom, Top), which makes the density at the center as 8 (4 times of that of the optimal solution), as illustrated in Figure 2.8. This example shows the tightness of the approximation ratio of our algorithm LPAPX.

### 2.4.4 Iterative Refinement

In this subsection, we present a greedy iterative refinement procedure as a postprocessing step to further improve the results obtained by the approximation algorithm LPAPX.

This greedy refinement procedure is performed iteratively. In each iteration, we try to re-escape all the rectangles one by one. When the re-escape for rectangle  $r_i$  is attempted, the escape directions of all the other rectangles are fixed. We try all the choices of escape directions for  $r_i$  and select the best one to be its new escape direction. The selection is done according to two criteria:

1. We pick the escape direction for  $r_i$  which results in smallest  $d_{max}$ .

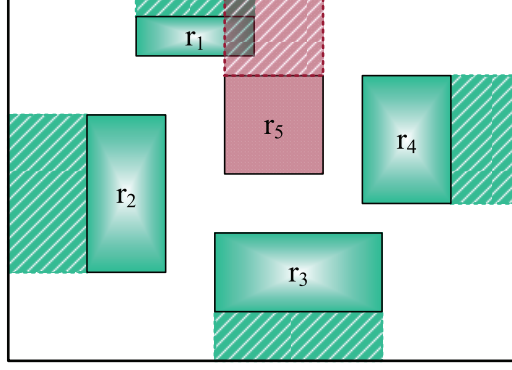


Figure 2.9: The rectangle  $r_5$  is set to escape to the top boundary since the least area is occupied in this way.

2. If two or more escape directions for  $r_i$  result in the same  $d_{max}$ , the area occupied by the rectangle after escaping is used to break the tie. We pick the direction that results in less occupied area after escaping, potentially leaving more room for other rectangles, which is probably beneficial for the refinement of other rectangles.

Figure 2.9 shows an example with 5 rectangles. Suppose we are re-escaping rectangle  $r_5$ , with the escape directions of the other rectangles fixed. It is easy to see that  $d_{max}$  will be 2 no matter how  $r_5$  escapes, but we decide to let it escape to the top boundary since the resultant rectangular region  $r_{5t}$  occupies the least area.

The pseudocode of the proposed iterative refinement procedure, GREEDYREFINE, is listed below.

```

ALGORITHM GREEDYREFINE(REP instance  $\mathcal{R}$ ):
    terminate  $\leftarrow$  false;
    while ! terminate do
        for each rectangle  $r_i \in S$  do
            Try all  $r_i$ 's escape directions;
            Pick the best one;
        if there is no improvement in this iteration then
            terminate  $\leftarrow$  true;
    return

```

### 2.4.5 Summary of the Algorithm

Our algorithm can be summarized as follows: Given a (weighted) REP instance  $\mathcal{R}$ , we first formulate it into an ILP, then apply LP relaxation and rounding technique to obtain an approximation solution, after which a greedy iterative refinement procedure is performed to improve the solution. The pseudocode of our algorithm flow, named **REPAPX**, is listed below.

ALGORITHM **REPAPX**((Weighted) REP instance  $\mathcal{R}$ ):  
 LPAPX( $\mathcal{R}$ );  
 GREEDYREFINE( $\mathcal{R}$ );  
**return**

## 2.5 Extensions

In this section, we show that our algorithm can be easily extended to handle more general versions of REP, namely, weighted REP and simultaneous REP.

### 2.5.1 Weighted REP

In general, a rectangle does not necessarily contribute a unit density, and the density one rectangle contributes may vary when it escapes in different directions. Hence in this general version of REP, each rectangle  $r_i \in S$  is associated with a weight vector  $w_i = [w_{il}, w_{ir}, w_{it}, w_{ib}]$ , where  $w_{il}$ ,  $w_{ir}$ ,  $w_{it}$  and  $w_{ib}$  denote the density rectangle  $r_i$  contributes when it escapes to the Left, Right, Top and Bottom, respectively. If two or more rectangles overlap with each other after escaping, the density of the overlapping region is calculated as the sum of the weights of the overlapping rectangles. We use weighted REP to denote this general version of REP.

#### **Weighted REP**

INSTANCE: A rectangular region  $R$  and a set  $S$  of  $n$  rectangles  $\{r_1, r_2, \dots, r_n\}$  residing within  $R$ . Each rectangle  $r_i$  is associated with a weight vector  $w_i = [w_{il}, w_{ir}, w_{it}, w_{ib}]$ , for  $1 \leq i \leq n$ .

QUESTION: Each rectangle  $r_i \in S$  chooses a direction to escape, such that  $d_{max}$  is minimized.

Similarly, a weighted REP instance  $\mathcal{R}$  can be formulated into the following ILP:

$$\begin{aligned}
& \text{Minimize} && d_{max} \\
& \text{Subject to} && \\
& && x_{il} + x_{ir} + x_{it} + x_{ib} = 1, && \forall i = 1, 2, \dots, n. \\
& && \sum_{i,* : r_{i*} \text{ occupies } p} w_{i*} \times x_{i*} \leq d_{max}, && \forall p \in P. \\
& && x_{il}, x_{ir}, x_{it}, x_{ib} \in \{0, 1\}, && \forall i = 1, 2, \dots, n.
\end{aligned}$$

This ILP can be relaxed into the following LP:

$$\begin{aligned}
& \text{Minimize} && d_{max} \\
& \text{Subject to} && \\
& && x_{il} + x_{ir} + x_{it} + x_{ib} = 1, && \forall i = 1, 2, \dots, n. \\
& && \sum_{i,* : r_{i*} \text{ occupies } p} w_{i*} \times x_{i*} \leq d_{max}, && \forall p \in P. \\
& && 0 \leq x_{il}, x_{ir}, x_{it}, x_{ib} \leq 1, && \forall i = 1, 2, \dots, n.
\end{aligned}$$

It is not difficult to figure out that the analysis in the proof of Theorem 2 still holds when we apply the algorithm LPAPX to a weighted REP instance. Thus we have the following theorem:

**Theorem 3.** *Given a weighted REP instance  $\mathcal{R}$ , where each rectangle has 4 candidate choices of escape directions, LPAPX is an 4-approximation algorithm for  $\mathcal{R}$ .*

*Proof.* Let us reuse the notations introduced in the proof of Theorem 2. We need to show that  $APX(d_{max}) \leq 4LP(d_{max})$ .

Similarly, after rounding, there must be a tile  $\hat{p} \in P$  such that the following equality holds:

$$\sum_{i,* : r_{i*} \text{ occupies } \hat{p}} w_{i*} \times \hat{x}_{i*} = APX(d_{max}). \quad \langle 4 \rangle$$

Note that the following two statements hold too:

- 1).  $\hat{x}_{i*} = 1 \leq 4x_{i*}$  if  $x_{i*}$  is a dominating variable ( $x_{i*} \in D$ ).
- 2).  $\hat{x}_{i*} = 0$  if  $x_{i*}$  is not a dominating variable ( $x_{i*} \notin D$ ).

Thus, we rewrite equation  $\langle 4 \rangle$  as follows:

$$\begin{aligned}
APX(d_{max}) &= \sum_{i,* : r_{i*} \text{ occupies } \hat{p}} w_{i*} \times \hat{x}_{i*} \\
&= \sum_{i,* : r_{i*} \text{ occupies } \hat{p}, x_{i*} \in D} w_{i*} \times \hat{x}_{i*} + \\
&\quad \sum_{i,* : r_{i*} \text{ occupies } \hat{p}, x_{i*} \notin D} w_{i*} \times \hat{x}_{i*} \\
&= \sum_{i,* : r_{i*} \text{ occupies } \hat{p}, x_{i*} \in D} w_{i*} \times \hat{x}_{i*} + 0 \\
&\leq \sum_{i,* : r_{i*} \text{ occupies } \hat{p}, x_{i*} \in D} w_{i*} \times 4x_{i*} \\
&\leq 4 \times \sum_{i,* : r_{i*} \text{ occupies } \hat{p}} w_{i*} \times x_{i*} \\
&\leq 4LP(d_{max}) \\
&\leq 4OPT(d_{max})
\end{aligned}$$

Therefore, LPAPX is a 4-approximation algorithm for a weighted REP instance.  $\square$

Similarly, we have the following immediate corollary:

**Corollary 2.** *Given a weighted REP instance  $\mathcal{R}$ , where each rectangle has  $\alpha$  candidate choices of escape directions, LPAPX is an  $\alpha$ -approximation algorithm for  $\mathcal{R}$ .*

In the PCB bus escape routing problem, it is possible that a bus occupies different numbers of layers for different escape directions. Take the bus in Figure 2.10 for example: One layer is enough to accommodate all the nets if the bus escapes to the left boundary or the right boundary, but two layers are needed if the bus escapes to the top boundary or the bottom boundary. This is due to the fact that the top boundary of the pin cluster's bounding box is much narrower than its right boundary. Weighted REP can exactly model this characteristic by the rectangle's weight vector. The weight vector of the rectangle for the bus in Figure 2.10 is  $[1, 1, 2, 2]$ .

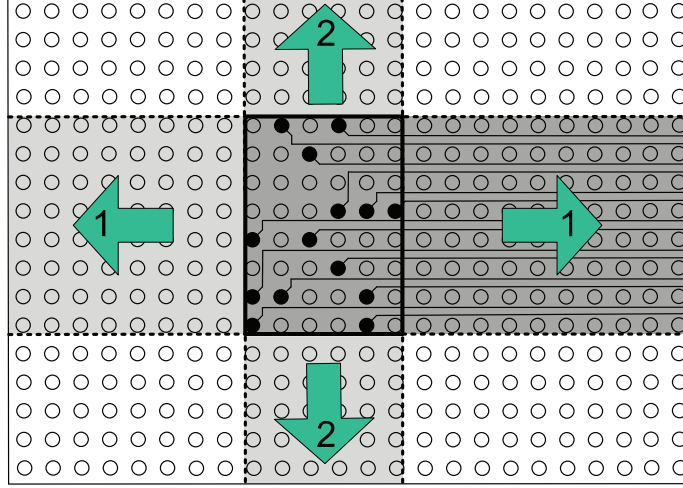


Figure 2.10: Bus escape routes in different directions may occupy different numbers of layers.

### 2.5.2 Simultaneous REP

Simultaneous REP models the simultaneous bus escape problem in PCB routing. It considers two components, each of which contains the same set of buses. The corresponding buses in the two components need to be connected. Figure 2.11 shows an example of simultaneous REP. The rectangle (representing a bus)  $r_1$  ( $r_2, r_3, r_4$ ) in the left component needs to be connected with the rectangle  $r'_1$  ( $r'_2, r'_3, r'_4$ ) in the right component. This imposes a set of constraints on the escape directions of rectangles in the two components:

- If rectangle  $r_i$  in the left component escapes to the right, the corresponding rectangle  $r'_i$  in the right component is preferred to escape to the left.
- If rectangle  $r_i$  in the left component escapes to the left, the corresponding rectangle  $r'_i$  in the right component is preferred to escape to the right.
- If rectangle  $r_i$  in the left component escapes to the top, the corresponding rectangle  $r'_i$  in the right component is preferred to escape to the top.
- If rectangle  $r_i$  in the left component escapes to the bottom, the corresponding rectangle  $r'_i$  in the right component is preferred to escape to the bottom.

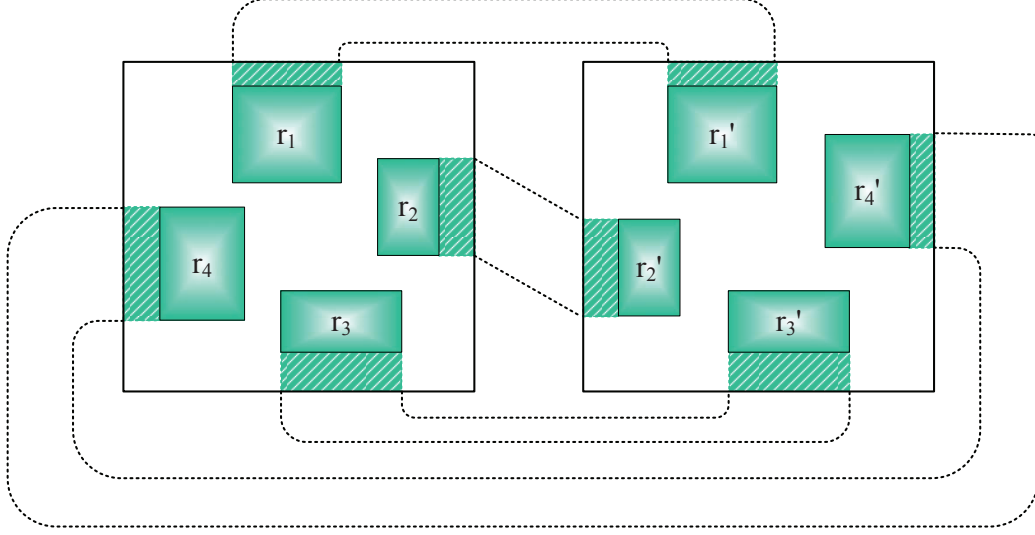


Figure 2.11: Simultaneous REP.

This is what we observed from industrial manual solutions of PCB bus escape routing, and satisfying these constraints usually makes things easier and more smooth for the length matching problem and the layer assignment problem in a later phase. Therefore, in simultaneous REP, we need to minimize the maximum density  $d_{max}$  over the two components, while satisfying the above constraints.

We show in the following that our linear programming relaxation and rounding technique can be easily extended and adapted to handle simultaneous REP.

Let  $\{x_{il}, x_{ir}, x_{it}, x_{ib}\}$  be the direction variables of rectangle  $r_i$  in the left component, and let  $\{x'_{il}, x'_{ir}, x'_{it}, x'_{ib}\}$  be the direction variables of the corresponding rectangle  $r'_i$  in the right component,  $\forall i = 1, 2, \dots, n$ . Simultaneous REP can be formulated into the following ILP:

$$\begin{aligned}
& \text{Minimize} && d_{max} \\
& \text{Subject to} && \\
& && x_{il} + x_{ir} + x_{it} + x_{ib} = 1, \quad \forall i = 1, 2, \dots, n. \\
& && x'_{il} + x'_{ir} + x'_{it} + x'_{ib} = 1, \quad \forall i = 1, 2, \dots, n. \\
& && \sum_{i,* : r_{i*} \text{ occupies } p} x_{i*} \leq d_{max}, \quad \forall p \in P. \\
& && \sum_{i,* : r'_{i*} \text{ occupies } p'} x'_{i*} \leq d_{max}, \quad \forall p' \in P'. \\
& && x_{il} = x'_{ir}, \quad \forall i = 1, 2, \dots, n. \\
& && x_{ir} = x'_{il}, \quad \forall i = 1, 2, \dots, n. \\
& && x_{it} = x'_{it}, \quad \forall i = 1, 2, \dots, n. \\
& && x_{ib} = x'_{ib}, \quad \forall i = 1, 2, \dots, n. \\
& && x_{il}, x_{ir}, x_{it}, x_{ib} \in \{0, 1\}, \quad \forall i = 1, 2, \dots, n. \\
& && x'_{il}, x'_{ir}, x'_{it}, x'_{ib} \in \{0, 1\}, \quad \forall i = 1, 2, \dots, n.
\end{aligned}$$

Similarly, we can relax the above ILP into LP and use an LP solver to solve it. We can then apply the same rounding technique to obtain an approximation solution. It is easy to figure out that the approximation ratio still holds. Therefore we have the following theorem, the detailed proof of which is omitted.

**Theorem 4.** *Given a simultaneous REP instance  $\mathcal{R}$ , where each rectangle has  $\alpha$  candidate choices of escape directions, LPAPX is an  $\alpha$ -approximation algorithm for  $\mathcal{R}$ .*

## 2.6 Experimental Results

We implemented our approximation algorithm REPAPX in C++, with *Gurobi Optimizer* [26] employed as our LP solver. To validate our proposed approach, we perform two sets of experiments on industrial PCB bus escape routing problems. The experiments are performed on a Linux workstation with two 3.0GHz Intel Xeon CPUs and 4GB memory.



Table 2.1: Comparison with ILP and greedy approach

Test Cases	# Bus	$d_{max}$					Runtime (sec)		
		REPAPX			ILP	Greedy	REPAPX	ILP	Greedy
		LP	APX	REP					
Ex1	16	1	1	1	1	1	0.012	0.063	0.008
Ex2	20	2	2	2	2	2	0.032	0.241	0.011
Ex3	24	2	2	2	2	2	0.016	0.078	0.015
Ex4	43	3.2	4	4	4	5	0.060	0.432	0.028
Ex5	44	3.08	4	4	4	5	0.064	0.671	0.032
Ex6	69	3	3	3	3	4	0.440	1.567	0.117
Ex7	106	3.14	4	4	4	5	1.328	51.79	0.329
Ex8	129	4.12	5	5	5	6	2.224	95.63	0.884
Ex9	148	4.4	6	5	5	6	2.640	118.6	1.263
Ex10	148	4.5	5	5	5	6	2.932	162.5	1.411
# OPT		10/10			10/10	3/10	-		

In the first set of experiments, we compare the performance of REPAPX with the ILP approach as well as a greedy approach on a set of unweighted REP test cases. In the ILP approach, the ILP is directly solved by the ILP solver (*Gurobi Optimizer*) without using LP relaxation. In the greedy approach, we directly apply the GREEDYREFINE procedure from the beginning. The experimental results are displayed in Table 2.1. There are 10 unweighted REP test cases, which are derived from industrial PCB bus escape routing data. The column “# Bus” indicates the number of Buses (the number of rectangles in the corresponding REP instance). The results obtained by our REPAPX algorithm are listed in the multi-column “REPAPX”. The “LP” column shows the  $d_{max}$  obtained after solving the LP (note that it might be a fractional number). The “APX” column shows the resultant  $d_{max}$  after rounding (LPAPX), while the “REP” column is the final maximum density  $d_{max}$  obtained after applying the iterative refinement procedure (GREEDYREFINE). The results obtained by the ILP approach and the greedy approach are listed in the “ILP” column and the “Greedy” column respectively. The ILP approach produces the optimal solution. We can see that our proposed approximation algorithm GREEDYREFINE obtains the same  $d_{max}$  value with the ILP approach for all the test cases, i.e., it optimally solves all the test cases, while the greedy approach only solves 3 test cases optimally. As for the runtime comparison, we can see that our approximation algorithm REPAPX is about 50x faster than directly solving the ILP for large test cases. Although the runtime of the ILP solver seems acceptable, solving ILP is NP-complete and there is no guarantee. When the problem size grows larger, the ILP solver can become unexpectedly slow. Figure 2.12 shows the bus escape

solution generated by REPAPX for test case *Ex2* with 20 rectangles, and the resultant maximum density  $d_{max} = 2$ . As we can see from the table, the gap between the results obtained by the greedy method and that by the proposed approach is not large. However, the greedy method can get stuck in some cases, due to its greedy nature. The example in Figure 2.13 gives an illustration. Figure 2.13(a) shows the solution obtained by the greedy approach. Suppose the escape directions of the buses are determined in the order of  $r_1, r_2, \dots, r_9$  in the greedy approach.  $r_2$  escapes to the top boundary, and  $r_4$  escapes to the right boundary. Then,  $r_6$  and  $r_8$  can only escape to the bottom boundary and left boundary respectively, since they will overlap with other buses otherwise. We will then notice that  $r_9$  will overlap with other buses no matter in which direction it escapes, so that we obtain an escape solution with  $d_{max} = 2$ . We then re-escape all the buses one by one, but it is easy to see that the greedy method gets stuck at this escape solution, as there is no improvement that can be obtained by changing the escape direction of the buses. Figure 2.13(b) shows the optimal escape solution with  $d_{max} = 1$ . It is possible that the greedy method can achieve the optimal solution by using a “good” ordering in which the buses are escaped. However, in general, finding a “good” ordering by itself is a very difficult problem (we cannot afford to enumerate all the possible orderings). Moreover, it is also possible that such a “good” ordering does not exist at all, i.e., no ordering gives us the optimal solution if the buses are escaped one by one (this is the nature of the greedy algorithm).

In the second set of experiments, we test our approximation algorithm REPAPX on a set of weighted REP test cases. For a bus, the weight associated with each escape direction is set to be the number of layers this bus occupies if it escapes to that direction. The results are shown in Table 2.2. Similar with that in Table 2.1, the columns “*LP*”, “*APX*” and “*REP*” show the  $d_{max}$  obtained after solving the LP, after rounding and after iterative refinement, respectively. The column “*OPT*” shows the optimal  $d_{max}$  computed by ILP. We can see that REPAPX solves each of the weighted REP test cases optimally within several seconds.

In each of the test cases, all the buses are allowed to escape in all the four directions. Although our algorithm REPAPX is a 4-approximation algorithm, the experimental results show that the performance of REPAPX is remarkably promising in practice, in the sense that each of the test cases

Table 2.2: Experimental results on weighted test cases

Test Cases	# Bus	REPAPX				
		$LP$ ( $d_{max}$ )	$APX$ ( $d_{max}$ )	$REP$ ( $d_{max}$ )	$OPT$ ( $d_{max}$ )	Runtime (sec)
Ex1w	16	2	2	2	2	0.016
Ex2w	20	3	3	3	3	0.042
Ex3w	24	3.3	4	4	4	0.435
Ex4w	43	4.2	5	5	5	0.117
Ex5w	44	4.55	6	5	5	0.120
Ex6w	69	5.12	7	6	6	1.262
Ex7w	106	6.34	7	7	7	3.781
Ex8w	129	7.16	9	8	8	5.526
Ex9w	148	7.32	9	8	8	5.903
Ex10w	148	7.45	8	8	8	6.650
# OPT		10/10				

(both the unweighted and weighted) can be optimally solved within several seconds.

## 2.7 Concluding Remarks

In this chapter, we introduce and study the Rectangle Escape Problem (REP), which originates in PCB bus escape routing. We prove that REP is NP-complete, and propose a 4-approximation algorithm by using linear programming relaxation and rounding technique. This algorithm is also shown to work for weighed REP and simultaneous REP. Our algorithm is implemented and tested on a set of industrial PCB bus escape routing cases, and the results show that an optimal solution can be obtained within several seconds for each test case, which confirms the efficiency and effectiveness of our proposed algorithm.

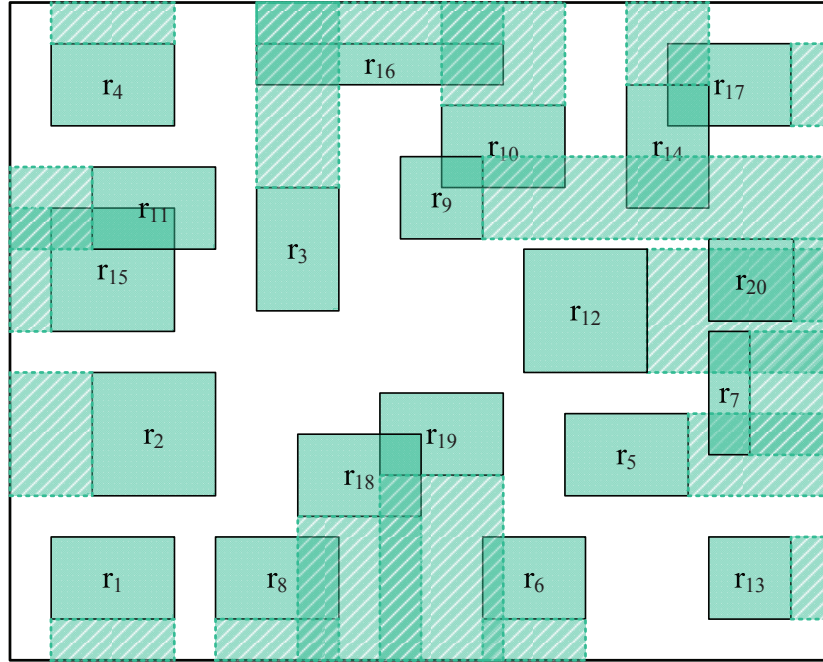


Figure 2.12: The bus escape solution generated by REPAPX for test case Ex2 with 20 rectangles ( $d_{max} = 2$ ).

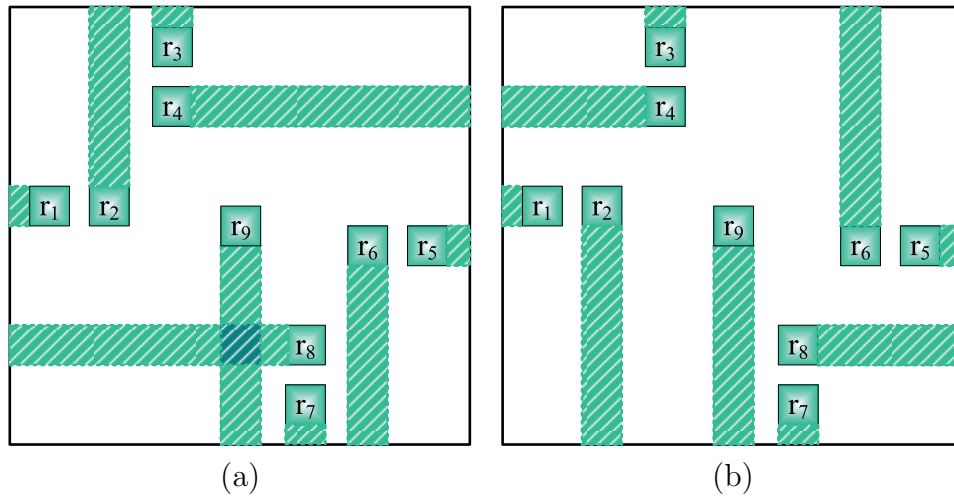


Figure 2.13: An example REP instance with 9 rectangles. The solution obtained by the greedy approach is shown in (a), while the optimal solution is shown in (b).

# CHAPTER 3

## NET-BASED ESCAPE ROUTING ON PCB

### 3.1 Introduction

Printed circuit board (PCB) routing is the problem of determining wiring connections between pin terminals on the circuit board. This problem has become extremely difficult due to the rapid increase in pin count and density, presence of differential pairs, and tight length-matching requirements. For today’s high-end complex PCBs, the routing problem can only be solved by a substantial amount of tedious and time-consuming manual effort. Thus, more research on design automation of PCB routing is greatly needed.

In this chapter, we focus on a key problem in PCB routing called escape routing. The objective of escape routing is to route all terminal pins inside pin grid (also called component) to the component boundaries, i.e., the pins are “escaping” from the components. Several types of escape routing problems have been studied in literature:

- **Single-component escape** considers routing all pins inside a component to the component boundary without any constraint on the pin ordering along the boundary [27–31].
- **Ordered escape** also considers only one component. However, it requires the escape routing to conform to specified ordering along the component boundary [32–34].
- **Simultaneous escape** considers escape routing of two components. The pin orderings of the escape routing for the two components are required to match each other in order to provide a planar topology for later detail routing between the components [35, 36].

The three types of escape routing problems have different applications in package and PCB routing. In this work, we consider the simultaneous escape

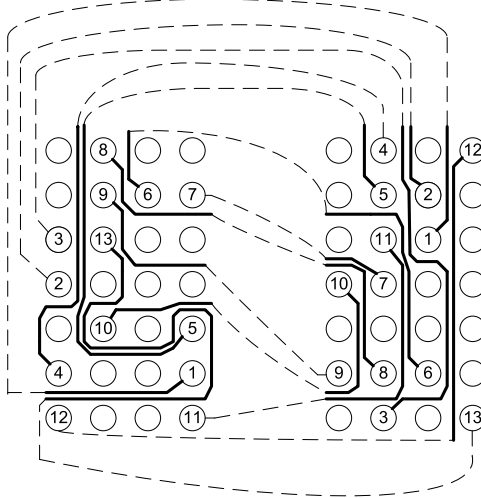


Figure 3.1: Simultaneous escape routing.

routing problem. Figure 3.1 gives an example of such a problem. Pins connecting two components are escaped to their respective component boundaries (the inter-component connections are shown in dotted lines and their detailed routing paths will be determined in a subsequent detailed routing phase).

The negotiated congestion based routing scheme is shown to be very effective on many routing problems, including FPGA routing [7], IC global routing [4–6] and even length-matching routing for PCB [37]. However, there have been no attempts to use it to solve the escape routing problem. In this chapter, we explore the possibility of applying this routing scheme to the simultaneous escape routing problem and test its effectiveness through experiments. In order to apply it to the escape routing, which is different from traditional IC routing and FPGA routing problems, we propose a routing graph to model the routing resources of the input pin grids. By applying the negotiated routing scheme on the constructed routing graph, we build a Negotiated Congestion based Escape Router (NCER). We compare the routability of our NCER with that of Allegro (a commercial PCB router from Cadence) on a set of industrial data. The experimental results show that NCER has routability comparable to that of Allegro. Moreover, their performances are complementary to each other, in the sense that most of the cases that cannot be completed by Allegro are completed by NCER. Therefore, NCER can be used as a supplement to Allegro: whenever Allegro fails, NCER can be tried to complete the routing.

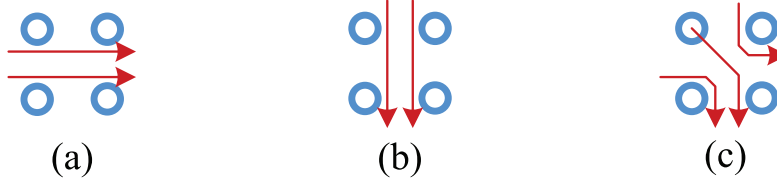


Figure 3.2: A routing tile of  $cap(2,2,3)$ : (a) horizontal capacity, (b) vertical capacity, (c) diagonal capacity.

The remainder of this chapter is organized as follows. We define the simultaneous escape routing problem in section 3.2. Then we discuss the construction of the underlying routing graph for the router in section 3.3. Section 3.4 reviews the negotiated congestion routing scheme and presents the details of our router. Section 3.5 describes another application of our NCER: bus untangling. Experimental results are reported in section 3.6 before conclusion and discussion in the last section. This work is published in [38].

## 3.2 Problem Formulation

On the pin grid of a PCB routing instance, a *routing tile* (*tile* for short) is a square region bounded by four neighboring pins. Each routing tile has a horizontal capacity, vertical capacity and a diagonal capacity, limiting the number of wires passing between orthogonally adjacent and diagonally adjacent pins. In most industrial PCB designs, a routing tile’s horizontal and vertical capacities are both 2, while the diagonal capacity can be 3 or 4. Figure 3.2 shows an example where the horizontal, vertical and diagonal capacities of a tile are respectively 2, 2 and 3. The notation  $cap(2, 2, 3)$  denotes the capacities of such a tile.

The simultaneous PCB escape routing problem can be stated as follows. The problem inputs are two components and the net number  $n$ , where each component is a rectangular pin grid, among which there are  $n$  pins labeled from 1 to  $n$ . The objective is to obtain detailed routing from these pins to the boundaries of the components so that ordering of the escaped pins along two components’ boundaries match each other (in order to provide a planar topology for inter-component connections). The escape routing inside the components is also required to satisfy the capacity constraints.

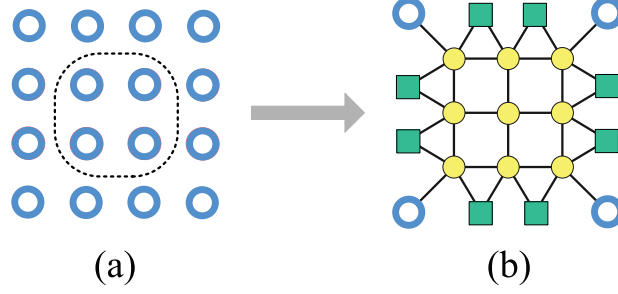


Figure 3.3: The graph model for one tile.

### 3.3 Underlying Routing Graph

In order to facilitate the negotiated congestion based routing, we need an underlying routing graph that correctly captures the planar property and capacity requirements. In this section, we present our modeling of the pin grids into a routing graph. We first propose a graph model for a single routing tile, then show the construction of the underlying routing graph by piecing together the tiles.

#### 3.3.1 Graph Model for a Tile

Figure 3.3 shows our graph model for one tile, which will be used as a unit structure of the routing graph. We first demonstrate that this unit structure correctly models a tile of  $cap(2, 2, 3)$ ; then we show that it can be adapted easily to model a tile of  $cap(2, 2, 4)$ .

In this graph model, the square nodes on the boundaries will be shared by neighboring tiles, and the circular nodes belong to this tile itself. Note that each node in this graph model can only accommodate one net, i.e., the node capacity is one. The square nodes are used to bound the horizontal and vertical capacities, while the circular nodes form the bottleneck of diagonal capacity. Figure 3.4 shows three distinct ways of routing nets through this graph model, where the horizontal, vertical and diagonal capacities are fully utilized, respectively. Theorem 5 shows the correctness of this graph model.

**Theorem 5.** *This graph model correctly captures the capacities of a routing tile of  $cap(2, 2, 3)$ .*

*Proof.* Take horizontal capacity for example. Since each node of the graph model can only accommodate one wire, it follows that the maximum number



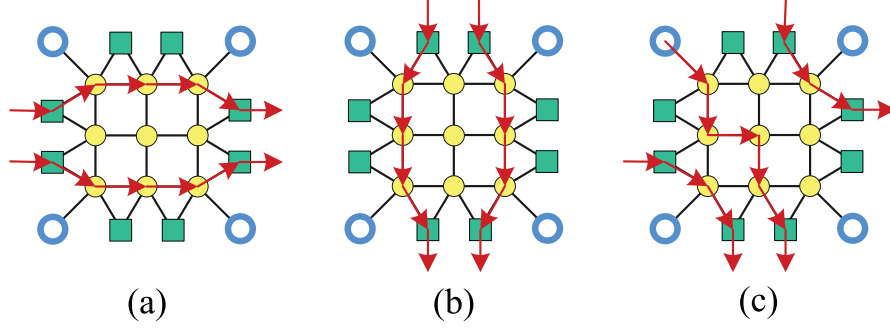


Figure 3.4: Illustration for the graph model's (a) horizontal capacity, (b) vertical capacity and (c) diagonal capacity.

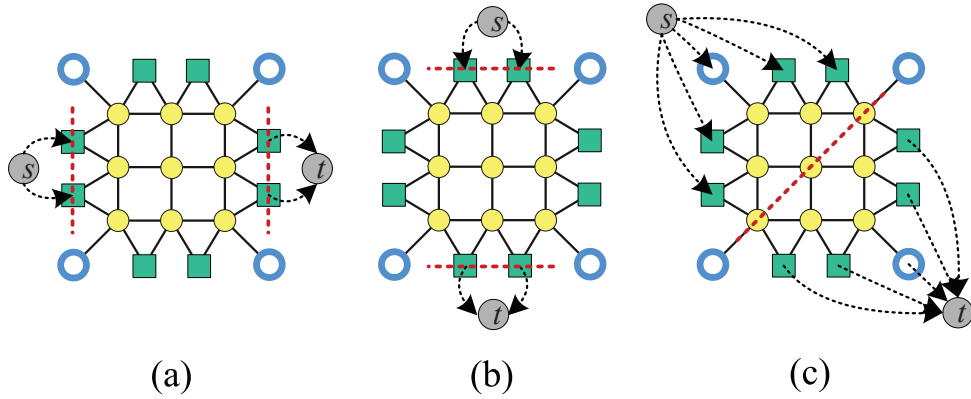


Figure 3.5: Proof of Theorem 5: (a) horizontal min-node-cut, (b) vertical min-node-cut, (c) diagonal min-node-cut.

of wires going through horizontally is the maximum number of horizontal node-disjoint-paths from the left boundary to the right boundary of the graph model, which is also the min-node-cut between source node  $s$  and sink node  $t$  if we connect  $s$  to the leftmost two square nodes and connect  $t$  to the rightmost two square nodes of the graph model, as shown in Figure 3.5(a). It is easy to see that the min-cut-node is of size 2, as marked by the dashed lines. Similar arguments can be applied to the vertical capacity and diagonal capacity, which completes the proof.  $\square$

The advantage of this graph model lies in that it can be easily modified to model the routing tile with  $cap(2, 2, 4)$ . By adding the four dashed edges as shown in Figure 3.6(a), the diagonal capacity of this graph model is augmented to four, without affecting the horizontal and vertical capacities. The correctness can be verified in the same manner as our proof of Theorem 5.

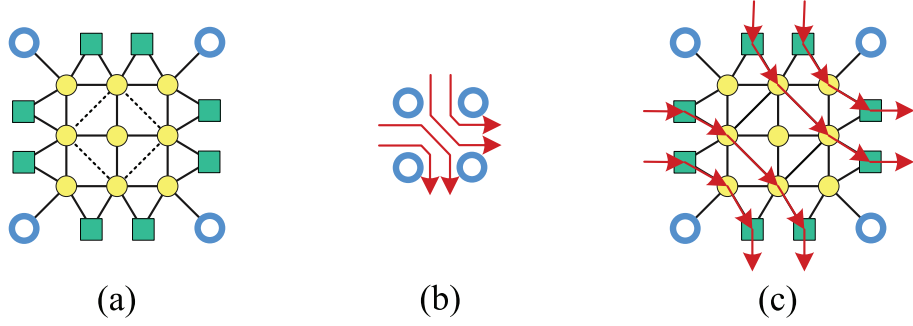


Figure 3.6: The graph model for a routing tile of  $cap(2,2,4)$ .

### 3.3.2 The Entire Routing Graph

The graph models constructed inside each tile are then pieced together to compose the routing graph of a component. Notice that the rectangular nodes in the figures are shared by adjacent tiles in a component. Therefore, the routing graph inside a component is connected. We also need to establish connections between the two components. In our implementation, we use a regular grid outside the components to connect them. Each boundary node of the component graphs is connected to a neighboring node in the outer grid graph, as illustrated in Figure 3.7. If the routing problem has  $n$  nets (i.e., each component has  $n$  pins to be escaped), then the outer grid graph has an  $n$  column/row margin around the components and an  $n$  column spacing between the two components. This margin and spacing guarantees the necessary routing resources between the boundaries of the two components. The following theorem shows that the routability of the original routing problem is preserved in our routing graph. The proof is omitted due to space limitation.

**Theorem 6.** *For any simultaneous escape routing instance, there exists a valid routing solution if and only if there exists a node-disjoint routing solution in the routing graph constructed.*

## 3.4 Negotiated Congestion

The negotiated congestion based routing scheme [7] has been widely used in FPGA routing and global routing, due to its balance of performance and

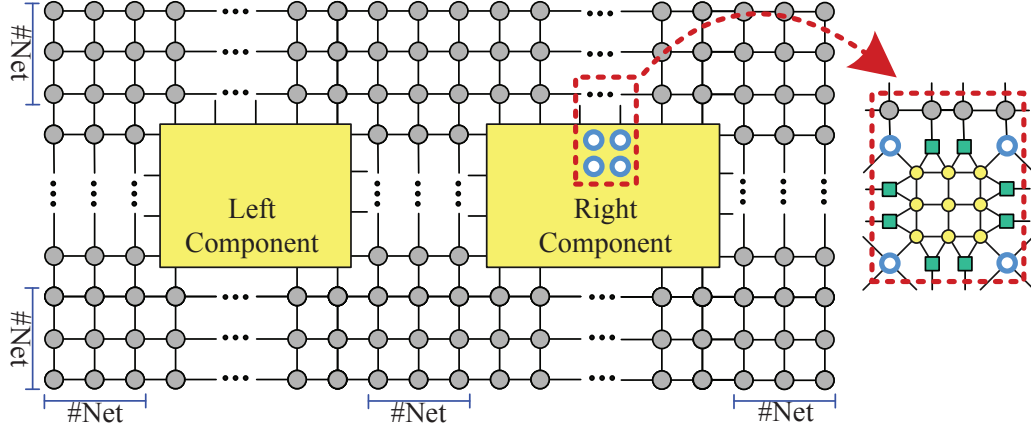


Figure 3.7: The whole routing graph.

routability. In this routing scheme, routability is achieved by forcing all the nets to negotiate for a resource and thereby determine which net needs the resource most. Some nets may use shared resources that are in high demand if all alternative routes utilize resources in even higher demand; other nets will tend to spread out and use resources in lower demand. All the nets are iteratively rerouted until no more resources are shared.

We apply this routing scheme on our routing graph to build our Negotiated Congestion based Escape Router (NCER). In the constructed routing graph  $G$ , the cost  $c_v$  of a node  $v$  is computed by the following formula:

$$c_v = b_v + h_v \times p_v$$

where  $b_v$  and  $h_v$  are respectively the base cost and history cost of node  $v$ , and  $p_v$  denotes the number of nets currently occupying node  $v$ . Our NCER works as follows. Before the routing starts,  $h_v$  and  $p_v$  are set to zero, so  $c_v = b_v$  for each node  $v$ . In the initial iteration, the  $n$  nets are routed one by one using the shortest path algorithm. Rip-up and reroute will then be performed if congestion exists, i.e., if there exists a node in the routing graph  $G$  that is occupied by more than one net. In each iteration of rip-up and reroute, every net is rerouted, even if the net does not pass through a congested area. In this way nets passing through uncongested areas can be diverted to make room for other nets currently in congested regions. At the beginning of each rip-up and reroute iteration, the history cost  $h_v$  of every currently congested node  $v$  is increased by  $\Delta$ . Note that  $\Delta$  increases with

iterations, which potentially causes nets to spread out from the congested areas more rapidly in subsequent iterations. The procedure terminates when all the routes are disjoint. The pseudocode of our NCER is listed below.

```

ALGORITHM NCER( $G, n$ ):
  for each node  $v$  in routing graph  $G$  do
     $h_v \leftarrow 0$ ;  $p_v \leftarrow 0$ ;  $c_v \leftarrow b_v$ ;
  Route all the  $n$  nets by shortest path algorithm;
  Update  $p_v$  and  $c_v$  for each node  $v$ ;
   $\Delta \leftarrow 1$ ;
  while  $\exists$  congestion do
    for each congested node  $v$  do
       $h_v \leftarrow h_v + \Delta$ ;
      Update  $c_v$ ;
    for  $i \leftarrow 1$  to  $n$  do
      Rip-up and reroute net  $i$ ;
      Update  $p_v$  and  $c_v$  of each affected node  $v$ ;
     $\Delta \leftarrow \Delta + 1$ ;

```

### 3.5 Application on Bus Untangling

Another application of NCER is the bus untangling problem studied in [39]. Given two columns of pins with labels  $\{1 \dots n\}$  on each column, the bus untangling problem is to detour the pins to make the ordering match in the middle (see Fig. 3.8 for an example). The detouring must also satisfy the capacity requirements between adjacent pins in each column. [39] provides an optimal algorithm for a single detour untangling problem, in which the detouring is only allowed in one column of pins. However, if we are allowed to detour on both sides, we can redistribute the occupied vertical tracks to two sides so that the maximum number of vertical tracks used on one side is reduced. Figure 3.8 gives an illustration of this observation. If we detour only on the left pin column (single detour), then we need to use two vertical tracks behind it. If the space behind the two pin columns is limited and can only accommodate one track, then this routing is invalid. If we are allowed to detour on both sides, we end up using one track on

each side, making the solution valid. Moreover, some inputs may not have single detour solutions. For example, if the left pin column is  $\{3, 4, 1, 2\}$  and the right column is  $\{1, 2, 3, 4\}$  from top to bottom, then there exist no single detour solutions. However, if we can detour on both sides, the above configuration has a solution.

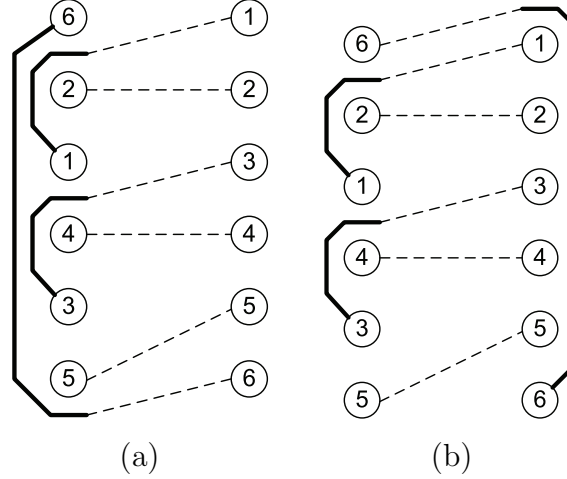


Figure 3.8: Bus untangling: (a) single detour untangling vs. (b) two sides untangling. Untangling on both sides reduces the maximum track usage on one side.

We can view the bus untangling problem as a special case of the escape routing problem, in which each pin column constitutes a component. We can apply our NCER to obtain a routing solution. Since NCER allows detours anywhere, the solution can have detours on both sides. Therefore, NCER produces solutions with less maximum track usage. It may also solve problems that cannot be solved by single detour untangling.

## 3.6 Experimental Results

We implemented our NCER in C++ and carried out the experiments on a Linux system with 2.8GHz CPU and 4GB memory. We tested NCER on 14 industrial test cases. Each of these problems takes approximately eight hours for an experienced layout engineer to manually complete the routing. Table 3.1 displays the detailed information about these test cases as well as performance comparison with Cadence PCB router Allegro. #Row and #Col denote the number of rows and columns of the input pin grids, and

Completion rate refers to the percentage of successfully routed nets in a test case.

From Table 3.1, we can see that that NCER and Allegro are comparable in terms of routability: both solved 7 out of the 14 test cases. Note that we use the total number of successfully routed cases as the measurement of routability instead of the completion rate of individual test cases. This is because completion rate does not reflect the actual rate of success. It is highly possible that a few unrouted pins are deeply buried inside the routes produced by an automated router and the only way to route those pins is to rip up and reroute most of the routed nets. Therefore, the number of completely routed cases is a better measurement of routability.

More interestingly, the performances of the two routers are complementary, in the sense that most of the test cases that cannot be accomplished by Allegro are 100% routed by NCER and most of the test cases that cannot be completed by NCER are completed by Allegro. Therefore, NCER can be used as a supplement to Allegro: whenever Allegro fails, NCER can be tried to obtain the routing. The runtime of NCER varies for different test cases (ranging from 20 to 6000 seconds for this set of data). It is difficult to compare NCER's runtime with that of Allegro, since Allegro is an integrated router that performs not only escape routing inside the components but also detailed routing between the components, and it is difficult to separate the escape routing time out of the total.

We do not have permission to display the routing solutions of the industrial test cases in Table 3.1, since they are confidential. We derived a simultaneous escape routing test case (with 45 nets) from these industrial data, and its routing solution generated by NCER is displayed in Figure 3.9.

We also applied NCER on two bus untangling problems. The first problem can be solved by the algorithm in [39], using 7 vertical tracks in the back of the left column. However, if we use NCER and detour on both sides, the maximum track number required on each side is only 4 (shown in Figure 3.10). The second problem can never be solved by the algorithm in [39], since it contains forbidden patterns for single detour untangling. However, NCER is able to solve it easily by detouring on both sides (see Figure 3.11).

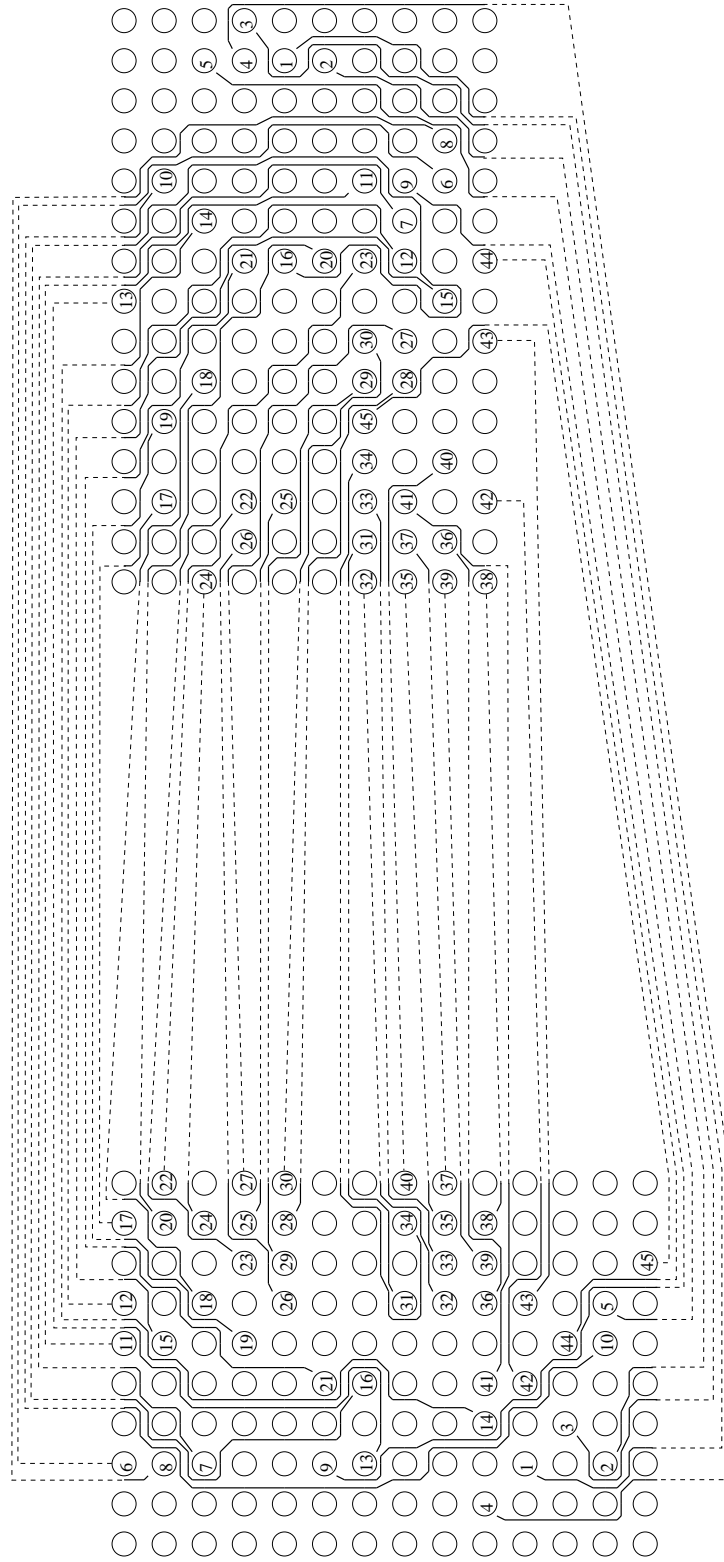


Figure 3.9: NCER's routing solution of a simultaneous escape routing problem with 45 nets.

Table 3.1: NCER vs. Cadence Allegro on industrial simultaneous escape routing test cases

Test Cases	#Nets	left pin grid #Row $\times$ #Col	right pin grid #Row $\times$ #Col	Completion rate	
				Allegro	NCER
Ex1	39	21 $\times$ 10	10 $\times$ 14	100%	79%
Ex2	36	14 $\times$ 9	20 $\times$ 8	100%	100%
Ex3	18	8 $\times$ 12	8 $\times$ 8	100%	100%
Ex4	26	13 $\times$ 16	13 $\times$ 13	95%	100%
Ex5	52	11 $\times$ 13	29 $\times$ 5	80%	77%
Ex6	40	7 $\times$ 7	10 $\times$ 8	100%	85%
Ex7	64	18 $\times$ 8	17 $\times$ 6	90%	78%
Ex8	32	16 $\times$ 6	16 $\times$ 6	100%	100%
Ex9	41	11 $\times$ 7	14 $\times$ 15	100%	93%
Ex10	37	8 $\times$ 14	11 $\times$ 5	95%	100%
Ex11	58	26 $\times$ 6	30 $\times$ 6	96%	100%
Ex12	36	16 $\times$ 14	9 $\times$ 13	100%	86%
Ex13	38	12 $\times$ 14	17 $\times$ 15	70%	100%
Ex14	39	18 $\times$ 18	15 $\times$ 13	80%	77%
# of routed problems				7/14	7/14

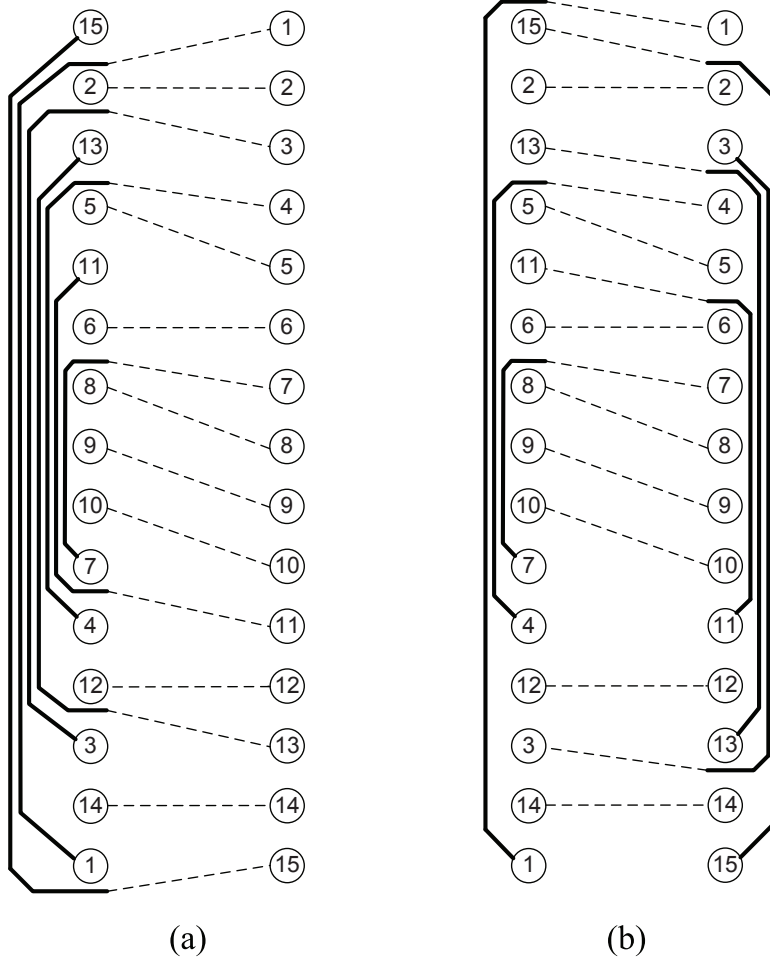


Figure 3.10: Applying NCER on bus untangling problem: result of NCER (b) has less maximum track usage than that of single detour untangling (a).



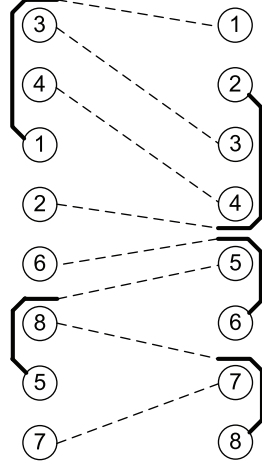


Figure 3.11: Applying NCER on bus untangling problem: NCER solves problem that cannot be solved by single detour untangling.

### 3.7 Conclusion

In this chapter, we study the feasibility of applying negotiated congestion routing scheme to simultaneous escape routing problems. Through experiments, we observe that the Negotiated Congestion based Escape Router (NCER) has performance comparable to that of the Cadence Allegro PCB router. Moreover, the two routers exhibit complementary behaviors. Combining them greatly improves the routability. This observation certainly opens more opportunities in the research of PCB routing. Successful routing techniques from other areas (like FPGA routing and IC routing) can be adapted to PCB routing despite its unique features (planar routing, strict length requirements, grid structure, etc.).

# CHAPTER 4

## LAYER ASSIGNMENT OF PCB ROUTING

### 4.1 Introduction

As the scale of modern electronic systems becomes larger and larger, the design of printed circuit boards (PCB) becomes more and more complex. Nowadays, a dense PCB hosts tens of thousands of pins. Manually routing such a large number of pins, which is what the industry has been doing, is a tedious and error-prone job. Therefore, design automation of PCB routing becomes a necessity.

The PCB routing problem is usually divided into two phases: (1) escape routing, which is to route nets from pin terminals to component (MCM, memory, etc.) boundaries (see the solid lines in Fig.4.1), and (2) area routing, which is to route nets between component boundaries (see the dashed line in Fig.4.1). In this chapter, we focus on the escape routing part. In practice, nets are usually grouped as buses and the nets from the same bus are expected to be routed together [17, 19, 40]. Directly applying net-centric escape routing algorithms may mix up the nets of different buses (as illustrated in Fig.4.1), which is not desired. It is observed from industrial manual routing solutions that the pins are usually escaped straight to the component boundaries with minimal detours. Therefore, the escape routes of all the pins of a bus are typically within one of its projection rectangles, which is obtained by projecting the bounding box of the bus pin cluster to one of the component boundaries [17, 19, 40]. Fig.4.2(a) demonstrates the concept of projection rectangle of a bus.

Due to the huge pin count and high density of the pin array, it usually requires multiple layers to escape the buses without any conflict. In fact, modern PCBs may contain more than 20 layers of routing [3]. The fabrication cost dramatically increases when more layers are needed, so we want to use as

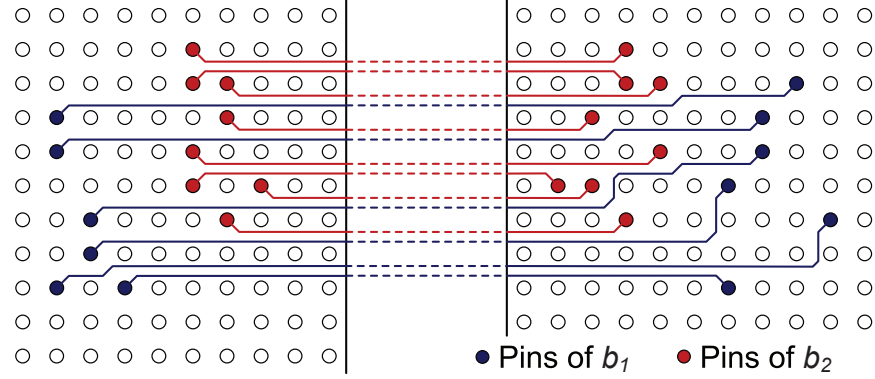


Figure 4.1: A sample net-centric escape routing solution for a problem with two buses. Nets of these two buses are mixed up.

few layers as possible to accommodate all the buses. Therefore, how to assign the escape routing of buses to different layers becomes an important issue. Another practical issue we observed is that a large bus is usually decomposed into two or more smaller buses, and these smaller buses are required to be routed on consecutive layers, as illustrated in Figure 4.2. Figure 4.2(b) is a large bus, which contains too many pins to be routed within its projection rectangle on one layer, so it is decomposed into two smaller buses, as shown in Figure 4.2(c) and Figure 4.2(d). The two resultant buses are able to be routed on one layer each; however, they are required to be placed on consecutive layers. This adds more constraints to the bus layer assignment problem.

Several previous works addressed this layer assignment problem for bus escape routing. Kong *et al.* proposed in [19] an optimal algorithm to determine if all the buses can be assigned to a single layer. Yan *et al.* in [40] optimally solved the layer assignment for multiple layers. However, these two works are based on the assumption that all the buses are escaped along the same boundary of a component, which greatly restricted the solution quality. Let us take the two buses  $b_1$  and  $b_2$  in Figure 4.1 as an example. Two layers are needed if both buses are routed to the right (left) boundary of the left (right) component, since otherwise they are mixed up. On the contrary, if bus  $b_1$  is escaped to the top boundary, the two buses can be routed on one layer without mixing, as shown in Figure 4.3. Recently in [17], an optimal algorithm is presented for finding a maximum independent set of bus projection rectangles escaping from one component on a single layer. The general problem

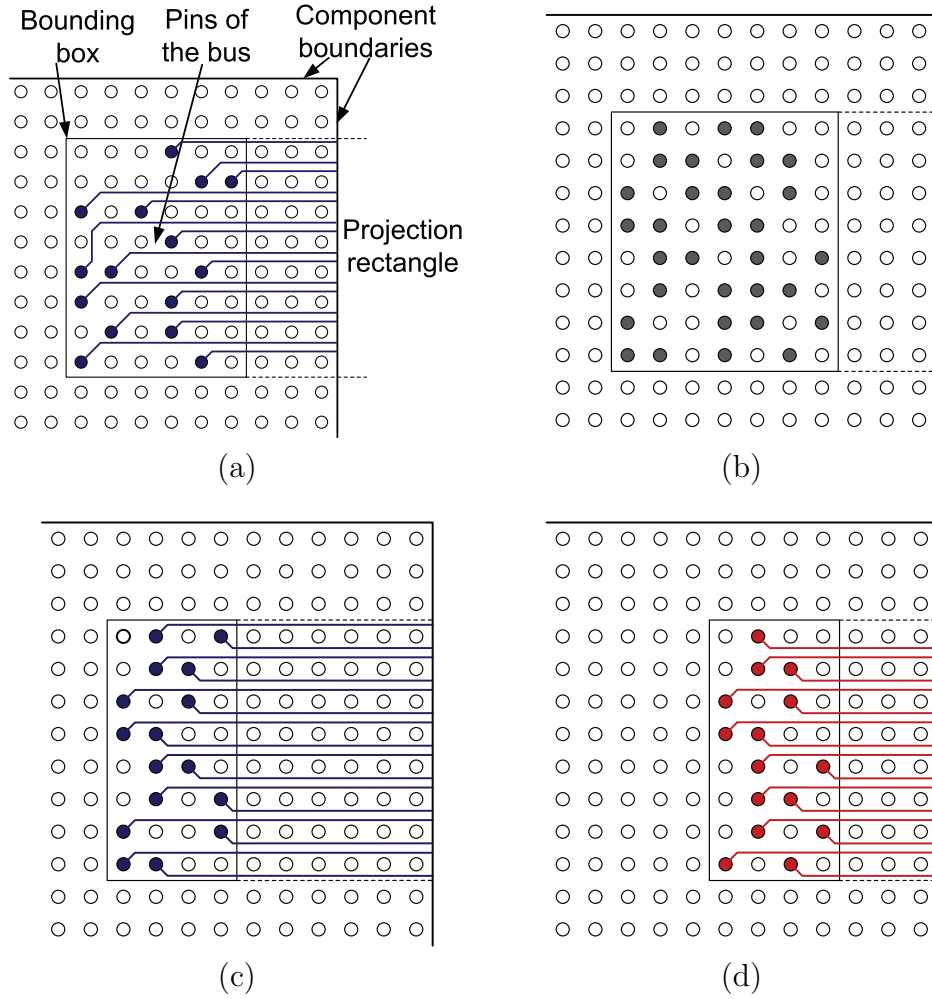


Figure 4.2: (a) illustrates the bus projection rectangle; (b), (c) and (d) show the bus decomposition issue: the large bus in (b) cannot be routed within its projection rectangle on one layer, so it is decomposed into two smaller buses ((c) and (d)), each of which is able to be routed on one layer. However, the two resultant buses are required to be routed on consecutive layers.

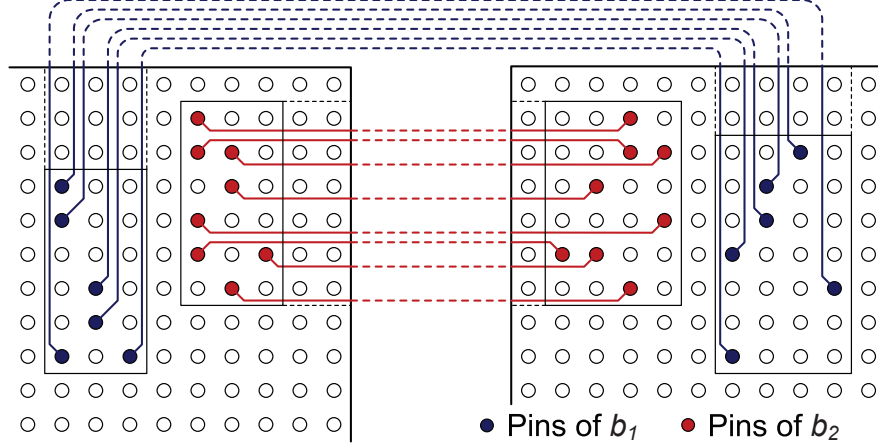


Figure 4.3: The two buses can be routed on one layer if bus  $b_1$  is escaped to the top boundary.

of optimal layer assignment of bus projection rectangles remains open and is the subject of this chapter.

Our contributions in this work are summarized as follows:

- We propose a branch-and-bound based algorithm, which optimally solves the problem of layer assignment for bus escape routing. The buses are allowed to be escaped to any of the component boundaries, and the constraints of consecutive layer assignment for buses generated after decomposition can also be properly taken care of during the searching.
- We observe through experiments that integer linear programming (ILP) is very efficient for finding maximum independent set and minimum coloring of bus projection rectangles, which makes it practical to integrate ILP into the branch-and-bound searching for upper and lower bounds computation.
- An effective min-cut based algorithm is developed to preprocess the bus projection rectangles, so that the search tree is constructed in such a way that early pruning is more possible, which further improves the searching efficiency.

The rest of this chapter is organized as follows: Section 4.2 defines the problem of layer assignment for bus escape routing; Section 4.3 presents our optimal algorithm for solving this problem; Section 4.4 reports the ex-

perimental results on some industrial data, and Section 4.5 concludes this chapter. This work is published in [41].

## 4.2 Problem Formulation

In our layer assignment problem of bus escape routing, we are given two components  $C_a$  and  $C_b$ , where  $C_a$  is located on the left and  $C_b$  is located on the right, facing each other. Other positions or orientations of the components can lead to similar discussions. We are also given a set  $B$  of  $n$  buses  $\{b_1, b_2, \dots, b_n\}$  connecting the two components, so we have  $n$  projection rectangles in each component representing the escape routing regions of the buses.

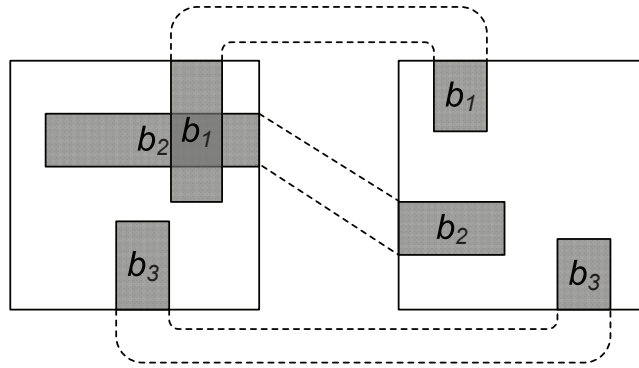
Some buses may have *internal conflict*, e.g.,  $b_1$  and  $b_2$  in Figure 4.4(a). We say that two buses have *internal conflict* if and only if their projection rectangles overlap in  $C_a$  or  $C_b$ . Some buses may have *external conflict*, e.g.,  $b_1$ ,  $b_2$  and  $b_3$  in Figure 4.4(b). We say that three or more buses have *external conflict* if and only if they cannot have a planar routing between the component boundaries (note that two buses can always be routed without external conflict). If a set of buses does not have either *internal conflict* or *external conflict*, we say that it is a *feasible set*. Figure 4.4(c) shows a *feasible set*.

Some buses come from the decomposition of a large bus, and they are required to be assigned to consecutive layers. We refer to this as a *consecutive assignment constraint*. In our problem input, we have a set of  $p$  consecutive assignment constraints,  $\{G_1, G_2, \dots, G_p\}$ , where  $G_i \subset B$ , and the buses in  $G_i$  have to be assigned to consecutive layers,  $\forall i \in \{1, 2, \dots, p\}$ .

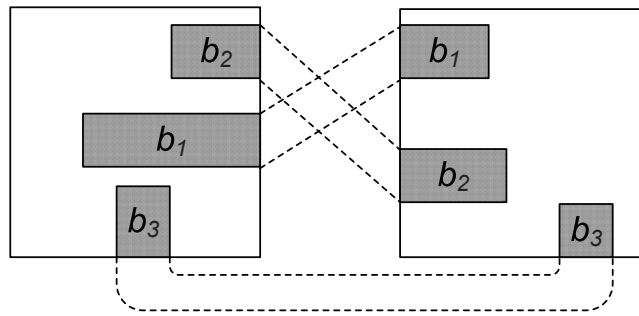
Our objective is to find a feasible layer assignment of all the buses with a minimum number of layers, satisfying the consecutive assignment constraints. We now formally define the layer assignment problem of bus escape routing as follows:

**Definition 2. Layer Assignment Problem of Bus Escape Routing -**

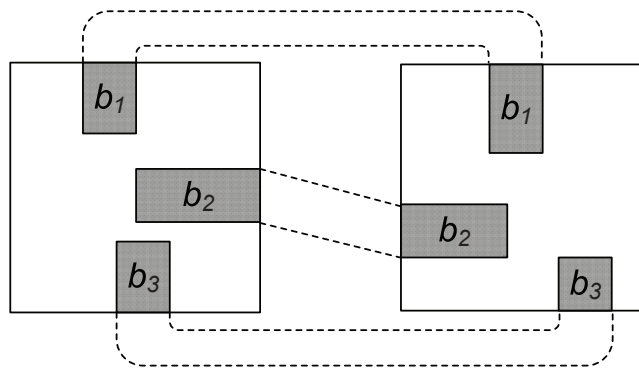
*We are given a set  $B$  of  $n$  buses  $\{b_1, b_2, \dots, b_n\}$  connecting between two components  $C_a$  and  $C_b$ , as well as a set of  $p$  consecutive assignment constraints  $\{G_1, G_2, \dots, G_p\}$ , where  $G_i \subset B$ ,  $\forall i \in \{1, 2, \dots, p\}$ . The objective is to assign all the buses to a minimum number of layers, such that the buses in*



(a)



(b)



(c)

Figure 4.4: (a) Internal conflict, (b) external conflict, (c) a feasible set.

*each layer is a feasible set, and the consecutive assignment constraints are satisfied.*

### 4.3 Optimal Layer Assignment

In this section, we present our optimal algorithm to solve the layer assignment problem. Our approach is based on branch-and-bound searching. We first give two algorithms which can efficiently compute an upper bound and a lower bound for the layer assignment problem, respectively; then we show how they are integrated into the searching process. Moreover, a min-cut based heuristic is developed to arrange the buses in a proper ordering in the search tree so that early pruning is encouraged, which helps improve the efficiency of the searching process.

#### 4.3.1 Upper Bound Computation

We first ignore the consecutive assignment constraints, and we will show later how to take those constraints into account during the branch-and-bound searching. An upper bound can be computed by finding a feasible layer assignment, i.e., the buses assigned to each layer form a feasible set. To compute a feasible set, we first find a set of buses without internal conflict, then resolve the external conflict if any exists. By iteratively finding a feasible set, a greedy method is developed to compute a feasible layer assignment.

##### Resolving Internal Conflict

Given a set of buses  $B$ , we first use ILP to compute a maximum set of buses  $S$  without internal conflict. Let  $b_i \perp b_j$  denote that bus  $b_i$  and bus  $b_j$  have internal conflict, and let  $x_i$  be a 0-1 variable indicating whether bus  $b_i \in B$  is selected into  $S$  or not. The following procedure ILP-MIS( $B$ ) returns the maximum set of buses  $S$  without internal conflict.



```

ALGORITHM ILP-MIS( $B$ ):
   $S \leftarrow \emptyset$ ;
  Solve the following ILP:
    Maximize  $\sum_{i:b_i \in B} x_i$ 
    Subject to
       $x_i + x_j \leq 1, \quad \text{if } b_i \perp b_j, \forall b_i, b_j \in B.$ 
       $x_i \in \{0, 1\}, \forall b_i \in B.$ 
  for each bus  $b_i \in B$  do
    if  $x_i = 1$  then
       $S \leftarrow S \cup \{b_i\}$ ;
  return  $S$ ;

```

We observe from experiments that this kind of ILP can be solved very efficiently by modern solvers. The chapter [17] gave an  $O(N^6)$  algorithm for computing the maximum independent set of bus projection rectangles within one component, where  $N$  is the total number of buses. We compare their algorithm with our ILP approach, and results show that our ILP approach can solve the problem much faster (see Section 4.4 for details).

#### Resolving External Conflict

The buses in  $S$  may still have external conflict, and we want to find from  $S$  a maximum subset  $S^*$  that has no external conflict. Figure 4.5 shows an example. We can obtain a sequence of the buses in  $S$  by traversing the boundaries of  $C_a$  clockwise (assume we start from its upper-left corner), and we call it a clockwise traversal of  $C_a$  (denoted by  $T_a$ ). Similarly, we can obtain another sequence of the buses in  $S$  by traversing the boundaries of  $C_b$  counter-clockwise (assume we start from its upper-right corner), and we call it a counter-clockwise traversal of  $C_b$  (denoted by  $T'_b$ ).  $T_a$  and  $T'_b$  in Figure 4.5 are respectively  $\langle b_1, b_2, b_3, b_4, b_5 \rangle$  and  $\langle b_3, b_4, b_2, b_5, b_1 \rangle$ . The following theorem can be easily observed.

**Theorem 7.** *A set of buses without internal conflict does not have external conflict if and only if  $T_a$  matches either  $T'_b$  or one of its rotations.*

Note that we need to consider the rotations, since the traversal is actually a circular ordering and the starting position should not be fixed (if  $T_a$  is

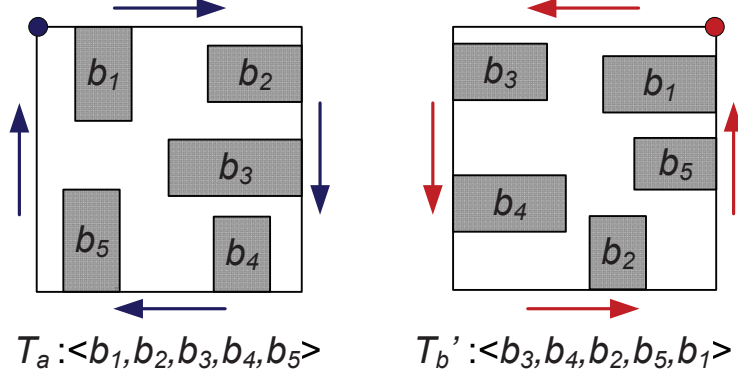


Figure 4.5: Buses without internal conflict have external conflict.

fixed, we need to check all the rotations of  $T_b'$ ). For example, in Figure 4.5, the rotations of  $T_b'$  include  $\langle b_4, b_2, b_5, b_1, b_3 \rangle$ ,  $\langle b_2, b_5, b_1, b_3, b_4 \rangle$ , and so on. We can assume  $T_b'$  itself is also one of its rotations. Based on Theorem 7, we can compute the longest common subsequences (LCS) of  $T_a$  and each rotation of  $T_b'$ , and the one with the maximum length gives the maximum subset  $S^*$  without external conflict. In Figure 4.5,  $T_a < b_1, b_2, b_3, b_4, b_5 >$  and one rotation of  $T_b' < b_1, b_3, b_4, b_2, b_5 >$  give a LCS of maximum length, which is  $\langle b_1, b_3, b_4, b_5 \rangle$ , so the maximum subset  $S^*$  without external conflict is  $\{b_1, b_3, b_4, b_5\}$ . Tang *et al.* showed in [42] that LCS can be computed in  $O(N \log \log N)$  time where  $N$  is the length of the input sequences. We now have an algorithm to compute a feasible set, which is shown below.

ALGORITHM FINDFEASIBLESET( $B$ ):

$S \leftarrow \text{ILP-MIS}(B)$ ;  
 Get the traversals  $T_a$  and  $T_b'$  of the buses in  $S$ ;  
 Compute the LCSs of  $T_a$  and each rotation of  $T_b'$ ;  
 Let  $\text{maxLCS}$  be the one with maximum length;  
 $S^* \leftarrow$  all the buses in  $\text{maxLCS}$ ;  
**return**  $S^*$ ;

#### Algorithm for Computing Upper Bound

Iteratively finding a feasible set gives us an upper bound on the optimal number of layers:

ALGORITHM CALUPPERBOUND( $B$ ):

```

upperbound  $\leftarrow$  0;
while  $B$  is not empty do
     $S^* \leftarrow \text{FINDFEASIBLESET}(B)$ ;
     $B \leftarrow B \setminus S^*$ ;
    upperbound  $\leftarrow$  upperbound + 1;
return upperbound;

```

### 4.3.2 Lower Bound Computation

Similarly, we first ignore the consecutive assignment constraints. Considering internal conflict and external conflict separately gives us two lower bounds on the optimal number of layers.

If we only consider internal conflict, the layer assignment problem becomes a minimum coloring problem of all the bus projection rectangles such that the buses assigned the same color do not overlap in  $C_a$  and  $C_b$ . Theoretically, it is a hard problem<sup>1</sup>. Again, we use ILP to solve this problem. Let  $\{x_{i1}, x_{i2}, \dots, x_{ik}\}$  be the set of 0-1 variables indicating whether bus  $b_i$  is assigned to layer 1, layer 2,  $\dots$ , or layer  $k$ , respectively. The following procedure can be used to check if there is a way to assign the buses into  $k$  layers without internal conflict. A lower bound can be obtained by performing a binary search on  $k$ .

ALGORITHM ILP-COLOR( $B, k$ ):

Check if there exists a solution to the following constraints:

$$\sum_{l=1}^k x_{il} = 1, \forall b_i \in B.$$

$$x_{il} + x_{jl} \leq 1, \forall l = 1, 2, \dots, k, \quad \text{if } b_i \perp b_j, \forall b_i, b_j \in B.$$

$$x_{i1}, x_{i2}, \dots, x_{ik} \in \{0, 1\}, \forall b_i \in B.$$

**if** *Yes* **then** **return** *true*;

**else** **return** *false*;

If we only consider external conflict, another lower bound can be obtained. We first compute the clockwise traversal  $T_a$  of  $C_a$  and also the clockwise

<sup>1</sup>The minimum coloring problem for bus projection rectangles is NP-complete even within one component, which can be proved by reduction from the circular arc coloring problem [43].

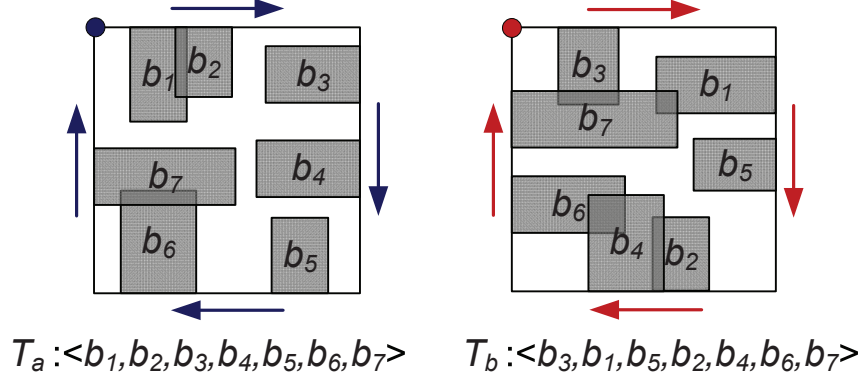


Figure 4.6: Clockwise traversal  $T_a$  and clockwise traversal  $T_b$ .

traversal  $T_b$  of  $C_b$ . Note that this time  $T_a$  and  $T_b$  contain all the buses of  $B$ . We then compute the LCSs of  $T_a$  and each rotation of  $T_b$ , and denote the one with maximum length by  $maxLCS^R$ , which means the reversed LCS is maximum length. It is easy to figure out that any three buses in  $maxLCS^R$  have external conflict, so they cannot be assigned to the same layer. As a result,  $\lceil \frac{|maxLCS^R|}{2} \rceil$  is a lower bound of the number of layers required.

Figure 4.6 gives an example of seven buses.  $T_a$  is  $\langle b_1, b_2, b_3, b_4, b_5, b_6, b_7 \rangle$ ,  $T_b$  is  $\langle b_3, b_1, b_5, b_2, b_4, b_6, b_7 \rangle$ , and  $maxLCS^R$  is computed as  $\langle b_1, b_2, b_4, b_6, b_7 \rangle$ . It is easy to check that any three of the five buses in  $maxLCS^R$  cannot be assigned to the same layer. Therefore,  $\lceil \frac{5}{2} \rceil = 3$  is a lower bound.

### 4.3.3 Branch-and-Bound Searching

In this section, we show how the layer assignment problem is solved optimally by using branch-and-bound searching. The consecutive assignment constraints can be properly taken care of during the searching. We describe the searching process by a search tree, in which an internal node represents a set of solutions while a leaf node represents either a single solution or an early termination of a searching branch.

#### Branching Rules

The root of the tree represents the set of all solutions (an optimal one exists for the original problem). We start from the root and visit different parts of

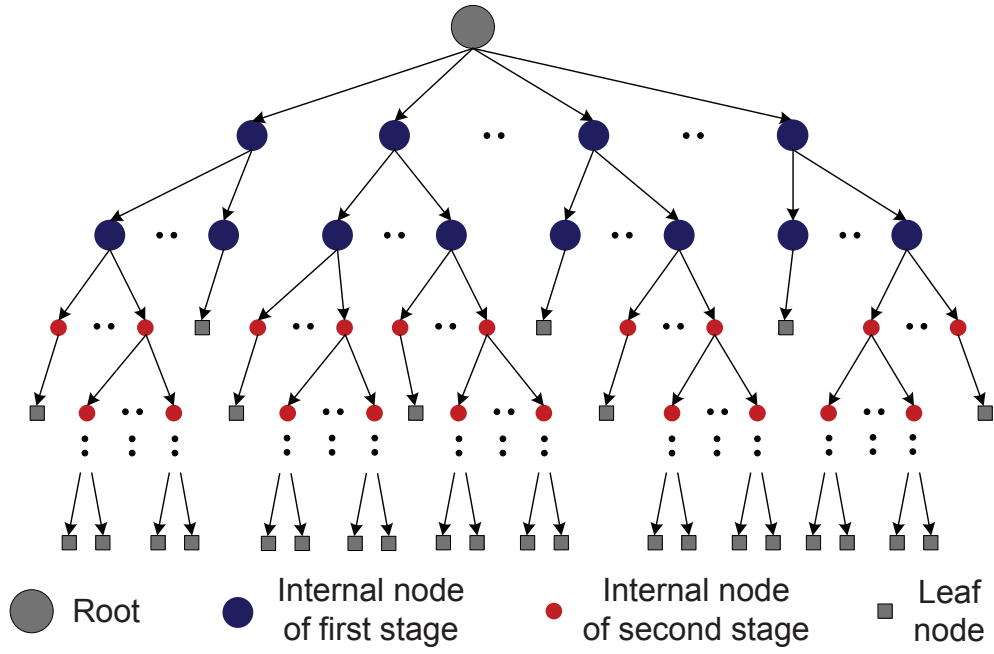


Figure 4.7: The search tree structure.

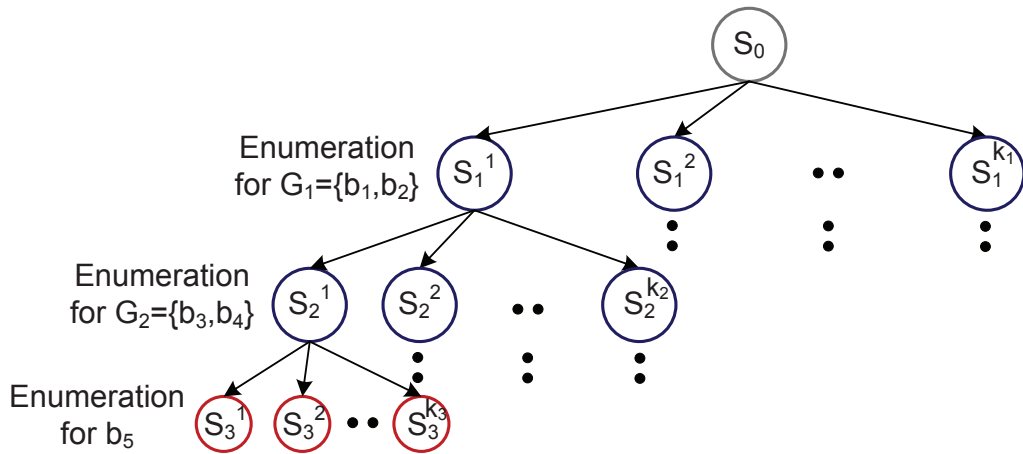


Figure 4.8: Sample of branch-and-bound method.

the tree by assigning certain buses to specific layers. There are two stages of our search tree, as shown in Figure 4.7. The first stage deals with the groups of buses with consecutive assignment constraints, where the nodes on each level enumerate all possible layer assignments of one group; the second stage handles the other buses without constraints, where the nodes on each level enumerate all possible layer assignments of one bus. We assume that the ordering of the groups (buses) in which we branch out have already been determined, and we will show in Section 4.3.4 how this ordering can be found such that early pruning is more possible.

The consecutive assignment constraints are handled in the first stage. These constraints are guaranteed to be satisfied by a systematical enumeration. Let us consider the following example. We have a set  $B$  of 5 buses  $\{b_1, b_2, b_3, b_4, b_5\}$  as well as two groups  $G_1 = \{b_1, b_2\}$  and  $G_2 = \{b_3, b_4\}$ . Let us assume, w.l.o.g., that the ordering for branching is set to be  $G_1, G_2, b_5$ . The sample search tree is shown in Figure 4.8, with node  $S_0$  as the root. In the first level, we enumerate all possible layer assignments for  $G_1$ , e.g., node  $S_1^1$  represents the set of solutions with  $b_1$  assigned to layer 1 and  $b_2$  assigned to layer 2 (while node  $S_1^2$  may represent the set of solutions with  $b_1$  assigned to layer 2 and  $b_2$  assigned to layer 1). At node  $S_1^1$ , we branch out to the next level, where the layer assignments for  $G_2$  are enumerated, e.g., node  $S_2^1$  represents the set of solutions with  $b_3$  assigned to layer 1 and  $b_4$  assigned to layer 2, in addition to the previous assignment of  $b_1$  and  $b_2$ . Now, there are no more groups with consecutive assignment constraints, so the branching process can enter the second stage, and the layer assignment of bus  $b_5$  is enumerated in the next level. In this way, we will branch out to many subproblems and search the whole solution space.

### Pruning Rules

In the search tree, some nodes with their subtrees together will never contribute an optimal solution, and good pruning rules help to find such nodes to reduce runtime. If a node is infeasible, that is, internal conflict or external conflict exists for the pre-assigned buses, it will be pruned. Besides, if a node's lower bound is no less than the global upper bound (denoted by  $O_{ub}$ ), it will also be pruned. This can be checked by the procedure `ILP-COLOR()` in Section 4.3.2. Note that additional constraints for the pre-assigned buses

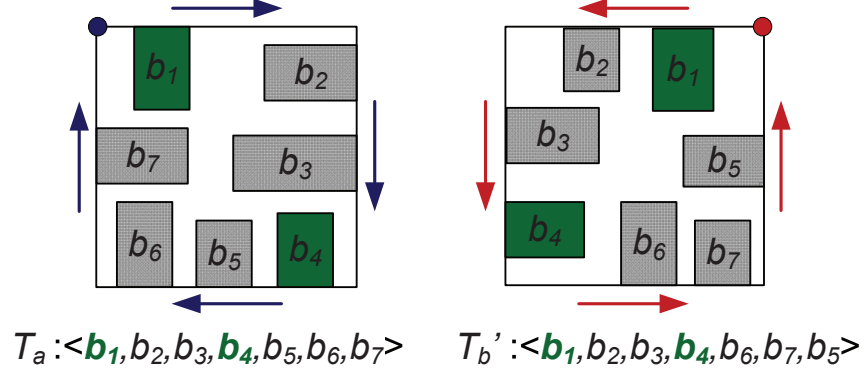


Figure 4.9: Computing LCS when there are pre-assigned buses.

need to be added into the ILP of  $\text{ILP-COLOR}()$  for this step. For example, at node  $S_1^1$  of Figure 4.8, bus  $b_1$  is assigned to layer 1 and bus  $b_2$  is assigned to layer 2, so the constraints  $x_{1,1} = 1$  and  $x_{2,2} = 1$  have to be added. The global upper bound  $O_{ub}$  is maintained during the searching, and it is updated whenever a better one is found. Note that the upper bound is only computed for the nodes where the buses with consecutive assignment constraints are all pre-assigned to some layers, i.e., an upper bound will not be computed for a node in the middle of the first stage. This is because our upper bound computation (the procedure  $\text{CALUPPERBOUND}()$  in Section 4.3.1) does not enforce the consecutive assignment constraints. From the end of the first stage, upper bounds are computed for the nodes. Similarly, additional constraints for the pre-assigned buses need to be added to the ILP of the procedure  $\text{ILP-MIS}()$  in Section 4.3.1. In addition, the pre-assigned buses also need to be addressed when LCS is computed between the two traversals. Basically, we compute the local LCSs between the pre-assigned buses and then concatenate them. Figure 4.9 shows an example where  $b_1$  and  $b_4$  are pre-assigned buses in this layer. The local LCS between  $b_1$  and  $b_4$  is  $\langle b_2, b_3 \rangle$ , and the local LCS after  $b_4$  is  $\langle b_6, b_7 \rangle$ , so concatenating them gives the LCS:  $\langle b_1, b_2, b_3, b_4, b_6, b_7 \rangle$ . With these bounds computation and pruning rules, we can search the solution space faster.

#### Target-Oriented Searching Rules

Our target-oriented branch-and-bound searching algorithm is divided into rounds. In each round, we set a target which is used as a threshold to decide

whether a node should be visited in this round. For example, in the first round, the target will be computed as:  $target = O_{lb} + C$ , where  $O_{lb}$  is the global lower bound of the original problem (the root in Figure 4.7), and  $C$  is a constant (set to 2 in our experiments). Actually,  $target$  is a guess we make on the optimal number of layers. If the lower bound of a node we reached is greater than  $target$ , this node will be marked and its subtree will not be visited in this round (may be visited in the following rounds). After one round, the global lower bound  $O_{lb}$  is updated as  $target + 1$ , and  $target$  will then be updated as  $target + C$ . Note that a subtree which is explored in a previous round will not be repeatedly visited in any subsequent rounds. In general, this target-oriented searching strategy will search nodes with smaller lower bounds in earlier rounds. This idea is derived from the branch-and-bound algorithm described in [44]. In our target-oriented branch-and-bound algorithm, there are three conditions under which we can stop searching: (1) A feasible solution is found at a node whose upper bound equals to the global lower bound; (2) Current round is finished and the global upper bound  $O_{ub}$  is less than or equal to  $target+1$ ; (3) All possible nodes are visited and searched.

Putting all the above steps together, we have the following pseudo-code for our Target-Oriented Branch-and-Bound (TOBB) algorithm.



```

ALGORITHM TOBB( $B, \{G_1, G_2, \dots, G_p\}$ ):
  Compute  $O_{lb}$  using ILP-COLOR() and binary search;
  Compute  $maxLCS^R$ ;
   $O_{lb} \leftarrow \lceil \frac{|maxLCS^R|}{2} \rceil$ , if  $\lceil \frac{|maxLCS^R|}{2} \rceil > O_{lb}$  ;
   $target \leftarrow O_{lb} + C$ ; // init  $target$ 
   $O_{ub} \leftarrow n$ ; // init  $O_{ub}$  as the number of buses
   $pool_{cur} \leftarrow \{\text{node } S_0\}$ ; // a queue of internal nodes
   $pool_{next} \leftarrow \emptyset$ ; // a queue of internal nodes
   $basket \leftarrow \emptyset$ ; // a stack of internal nodes
  repeat
    while  $pool_{cur}$  is not empty do
       $S \leftarrow$  get a node from  $pool_{cur}$ ;
      Push  $S$  into  $basket$ ;
      while  $basket$  is not empty do
         $S \leftarrow$  pop a node from  $basket$ ;
        if conflict exists for pre-assigned buses in  $S$  then
          continue; // this subtree is pruned
        if ! ILP-COLOR( $B, O_{ub} - 1, S$ ) then
          continue; // this subtree is pruned
        if ! ILP-COLOR( $B, target, S$ ) then
          Put  $S$  into  $pool_{next}$ ; //  $S$  is not visited now
          continue;
        if first stage finished then
           $S_{ub} \leftarrow \text{CALUPPERBOUND}(B, S)$ ;
          if  $S_{ub} = O_{lb}$  then return  $S_{ub}$ ;
          if  $S_{ub} < O_{ub}$  then  $O_{ub} \leftarrow S_{ub}$ ;
          Generate and push all children of  $S$  into  $basket$ ;
      if  $O_{ub} \leq target + 1$  then return  $O_{ub}$ ;
      if  $pool_{next}$  is empty then return  $O_{ub}$ ;
       $pool_{cur} \leftarrow pool_{next}$ ;  $pool_{next} \leftarrow \emptyset$ ;
       $O_{lb} \leftarrow target + 1$ ;  $target \leftarrow target + C$ ;

```

#### 4.3.4 Min-Cut Based Bus Ordering Algorithm

The ordering of the buses in the search tree affects the searching efficiency. For example, two buses  $b_i$  and  $b_j$  overlap with each other, so they cannot be assigned to the same layer. Suppose bus  $b_i$  is assigned to a specific layer at some internal node  $S$ , and the subtree rooted at  $S$  is being searched. After going down the tree many levels by enumerating the layer assignments for some other buses, we arrive at the level for bus  $b_j$ . When we try to assign  $b_j$  to the same layer with  $b_i$ , we realize that conflict exists and this searching direction should be terminated. Actually, this searching direction could have been terminated much earlier if  $b_j$  were immediately after  $b_i$  in the bus ordering, so that the solution space could be explored much faster. Therefore, we develop an algorithm to determine the bus ordering for searching efficiency.

Intuitively, we want to construct the search tree in a way that the subtrees containing infeasible solutions are pruned as early as possible. It is observed that early pruning is favored when the buses with more conflict with each other stay closer in the bus ordering. Based on this observation, we develop our min-cut based bus ordering algorithm, which is described below with a concrete example.

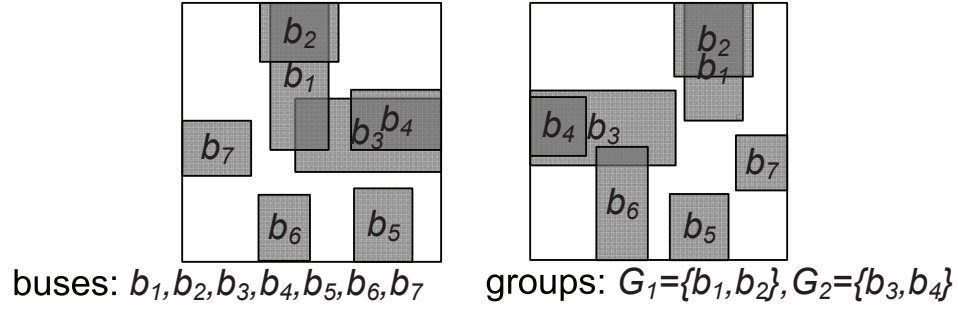
- Build a correlation graph of the buses. This graph depicts the conflict information of all the buses. Each vertex represents a bus. For every two buses with internal conflict, we add an edge with weight  $w_{in}$  between the two corresponding vertices. For every three buses with external conflict (and without internal conflict), we add an edge with weight  $w_{ex}$  between each two of the three corresponding vertices. In our implementation,  $w_{in}$  is set to 10 and  $w_{ex}$  is set to 1. Figure 4.10(a) shows a problem instance with 7 buses and 2 groups, and Figure 4.10(b) is the correlation graph constructed.
- Merge the vertices belonging to the same group. The edges connecting a vertex outside the group and the vertices inside the group are also merged, and their weights are summed up. Figure 4.10(c) shows the graph after merging. The groups need to be placed at the head of the ordering (in the upper level of the search tree), and the other buses are placed at the tail (in the lower level of the search tree), so a partial ordering is found and now the graph becomes two separate subgraphs.

The partial ordering in Figure 4.10(c) is  $\{G_1, G_2\}, \{b_5, b_6, b_7\}$ .

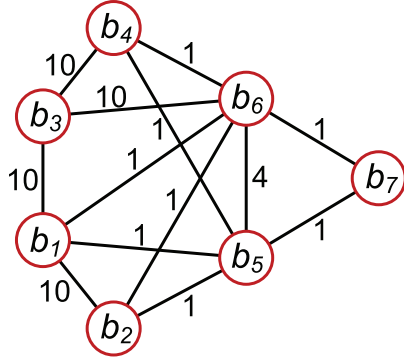
- We then iteratively apply the min-cut algorithm on both subgraphs. Each subgraph further divides into two smaller subgraphs after cutting, and a finer partial ordering is obtained. Our strategy for deciding the relative ordering of the two smaller subgraphs is as follows: the one with more conflict with the buses in front is put closer to the head of the ordering, so that potentially more pruning can be done earlier in the search. In Figure 4.10(d),  $\{G_1, G_2\}$  is cut into  $G_1$  and  $G_2$ , and  $\{b_5, b_6, b_7\}$  is cut into  $\{b_5, b_6\}$  and  $b_7$ . The new partial ordering is  $G_1, G_2, \{b_5, b_6\}, b_7$ . Since  $\{b_5, b_6\}$  has more conflict with  $\{G_1, G_2\}$ , it is put in front of  $b_7$  in the ordering. The min-cut algorithm is iteratively applied until each subgraph contains only one vertex, and a complete ordering is decided. In Figure 4.10(e),  $\{b_5, b_6\}$  is cut into  $b_5$  and  $b_6$ , and  $b_6$  is placed in front of  $b_5$  since  $b_6$  has more conflict with the buses in front. Thus, a complete ordering is obtained. The min-cut of an undirected edge-weighted graph  $G(V, E)$  can be computed in time  $O(|V||E| + |V|^2 \log |V|)$  [45].

## 4.4 Experimental Results

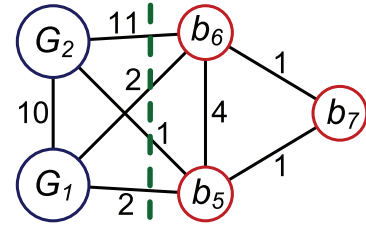
We implemented our algorithms in C++, with Gurobi Optimizer [26] employed as our ILP solver. We first test using ILP to find maximum independent set (MIS) and minimum coloring (MC) of bus projection rectangles. The results are shown in Table 4.1. The test cases are the same as those in [17], and each test case contains only one component. Results show that ILP is quite efficient in solving MIS and MC, making it practical to be embedded into branch-and-bound search. We can see that ILP solves MIS in much shorter time compared with the optimal algorithm proposed in [17]. We then test our branch-and-bound algorithm on a set of test cases derived from industrial data. Each test case contains a set of buses connecting two components, and some of them have consecutive assignment constraints. The results are displayed in Table 4.2. The column “# Bus” and the column “# Group” show the number of buses and the number of groups with consecutive assignment constraints, respectively. The column “# Net” indicates the



(a)

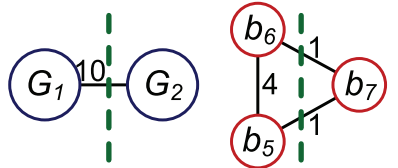


(b)



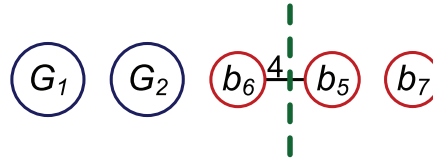
Ordering:  $\{G_1, G_2\}, \{b_5, b_6, b_7\}$

(c)



Ordering:  $G_1, G_2, \{b_5, b_6\}, b_7$

(d)



Ordering:  $G_1, G_2, b_6, b_5, b_7$

(e)

Figure 4.10: (a) A bus escape routing instance. (b) The correlation graph constructed. (c) The correlation graph after merging; the vertices representing groups and the ones representing other buses are separated. (d) Applying the min-cut algorithm. (e) Applying the min-cut algorithm again.

Table 4.1: Runtime of ILP on MIS and MC

Test Cases	# Bus	# Net	Runtime of MIS (sec)		Runtime of MC (sec)
			ILP	[17]	
Ex1	8	272	0.003	0.005	0.010
Ex2	8	680	0.004	0.005	0.011
Ex3	11	712	0.004	0.007	0.016
Ex4	13	286	0.005	0.011	0.015
Ex5	14	644	0.005	0.015	0.016
Ex6	43	1539	0.012	0.272	0.423
Ex7	44	1428	0.012	0.320	0.439
Ex8	64	762	0.028	1.320	1.024

Table 4.2: Results of our branch-and-bound algorithm

Test Cases	# Bus	# Group	# Net	Results	
				Opt. # Layer	Runtime (sec)
bb10	10	0	358	3	0.31
bb12	12	2	168	4	2.27
bb20	20	3	484	4	36.8
bb30	30	3	610	6	4.53
bb40a	40	0	432	10	932
bb40b	40	8	588	10	4299
bb50	50	6	946	7	546
bb60	60	8	1156	6	1842

total number of nets contained in the buses for each test case. The column “Results” reports the optimal number of layers and the runtime. The experiments are performed on a Linux workstation with two 3.0GHz Intel Xeon CPUs and 4GB memory.

## 4.5 Concluding Remarks

In this chapter, we propose a novel branch-and-bound based optimal algorithm for the layer assignment problem of bus escape routing on PCBs. The practical consecutive assignment constraints are addressed and handled. This is the first optimal algorithm proposed in the literature for this problem. Experimental results on industrial data validate our approach.

# CHAPTER 5

## ROUTING WITH GRAPHENE NANORIBBONS

### 5.1 Introduction

The semiconductor industry has showcased a spectacular exponential growth in the number of transistors per integrated circuit for several decades, as predicted by Moore’s law. However, maintaining this exponential growth rate in the future is a major challenge. Conventional CMOS devices are facing a growing number of issues as feature sizes scale down, including increased wire resistivity due to surface and grain-boundary scattering, increased leakage power, significant mobility degradation, and large dopant fluctuations. Chemically synthesized nanoscale materials—such as nanowires, carbon nanotubes (CNTs), and graphene nanoribbons (GNRs)—have been shown to have favorable device properties, new device characteristics, and large integration capability through new fabrication techniques. These nanoscale devices have significant potential to revolutionize the fabrication and integration of electronic systems and operate beyond the perceived scaling limitations of traditional CMOS.

Recently, a significant number of studies have focused on building field-effect transistors (FETs) using CNTs and GNRs. CNFETs are field-effect devices that use CNTs for their channels, while GNRFETs use thin ribbons of graphene as the channel material. The primary advantage of GNRFETs over CNFETs is the two-dimensional structure of graphene. Since graphene is created in large homogeneous sheets, it can be grown and patterned using standard planar processing technology. This makes it easier to work with than nanotubes, which require a bottom-up method of fabrication in which the tubes must be aligned and placed either during growth or in a subsequent processing step. Recently, top-gated GNRFETs of various gate lengths have been fabricated with peak cutoff frequencies up to 26 GHz for a 150nm gate

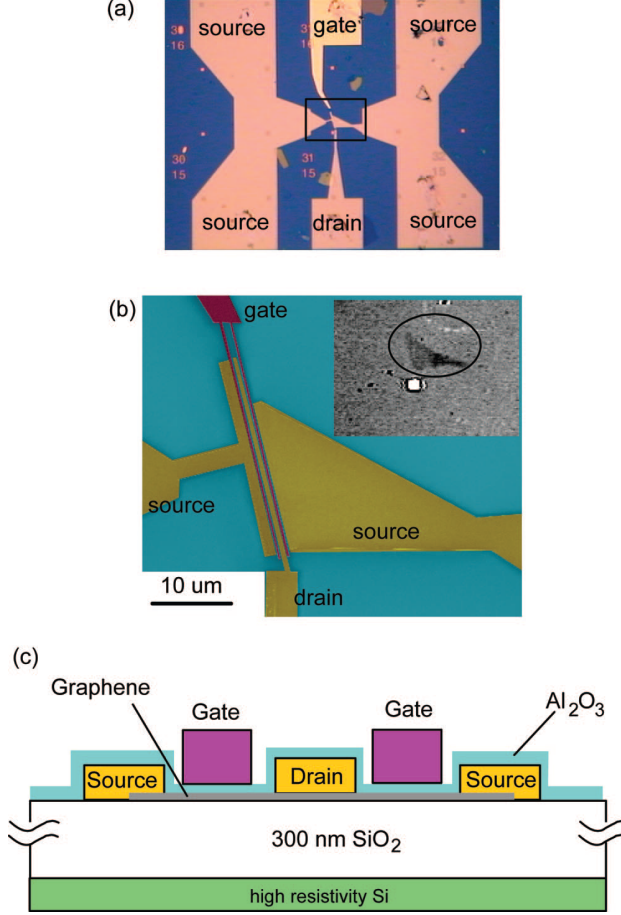


Figure 5.1: Layout of a GNRFET design. (a) Optical image of the device layout. (b) Scanning electron microscopy image of the graphene channel and contacts. (c) Schematic cross section of the graphene transistor.

length [46] (see Figure 5.1). Results indicate that if the high mobility of graphene can be preserved during the fabrication process, a cutoff frequency approaching terahertz may be achieved for GNRFET with a gate length of 50nm [8, 46].

While previous innovations in GNR electronics originated at the individual device level, realizing the true impact on electronic systems demands that we translate these device-level capabilities into system-level benefits. This translation requires new design automation algorithms for GNR nanocircuits that honor the unique constraints imposed by GNRs and target the key advantages they offer. Design automation has always kept pace with the advance of semiconductor technology. Research in design automation usually starts even before the technology is actually put in use. Given the great

potential of GNR-based nanoelectronics, it is important to start research on the related design automation issues. This chapter focuses on the GNR routing problem.

Due to the unique benzene-like hexagonal ring structure of graphene, metallic zig-zag GNRs can be of three different orientations, implying a routing grid very different from the rectangular grid for conventional metal routing. Furthermore, the delay of a metallic GNR wire is determined not only by its length but also by the number of bendings and the angle of bendings along its path. Obviously, the maze routing algorithm pervasively used in traditional IC routers is not applicable to such a routing problem. We need a new routing model to capture the unique routing grid and delay metric.

In this chapter, we propose to formulate the single-net GNR routing problem as a minimum hybrid-cost shortest path problem on a triangular mesh (“hybrid” means that both the length and the bendings are integrated together as the cost). This problem is then modelled as a shortest red-black path problem by expanding the graph. We then perform another graph expansion and show that by applying classical shortest path algorithm, i.e., Dijkstra’s algorithm on the expanded graph, the shortest red-black path problem can be optimally solved. This leads to an optimal algorithm for the minimum bending-cost shortest path problem. The algorithm is then fitted into a negotiated congestion based routing scheme to solve a multi-net problem. This is the first time that the GNR routing problem is formulated and studied.

The rest of this chapter is organized as follows: Section 5.2 introduces the necessary background of graphene nanoribbons. Section 5.3 formulates the GNR routing problem. The problem is then optimally solved in Section 5.4. Section 5.5 gives the negotiated congestion based routing scheme used to address multi-net routing. Section 5.6 shows some preliminary experimental results and Section 5.7 concludes the chapter. This work is published in [47].

## 5.2 Background

Carbon nanomaterials are composed primarily of benzene-like hexagonal rings of carbon atoms. Each edge of the hexagon is an  $sp^2$  carbon-carbon bond with a bond length of roughly 0.14nm. The intrinsic physical and



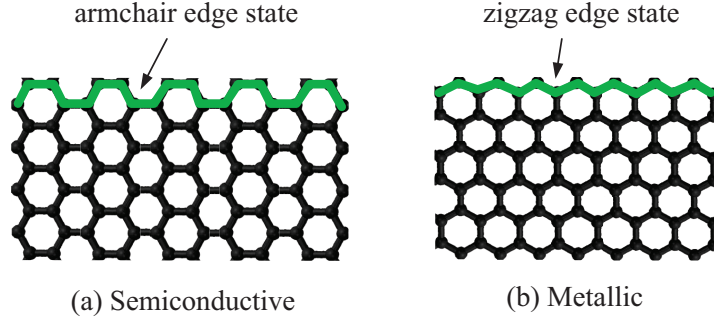


Figure 5.2: Armchair edge states lead to semiconductive GNRs while zigzag edge states lead to metallic GNRs.

electrical properties of graphene make it desirable for a large number of potential applications ranging from biosensors to flexible electronics to solar cell electrodes. One of the most exciting applications is the use of graphene in future high performance transistors. GNRs offer a mean free path 25x longer than copper, and carrier mobilities over 10x higher than silicon [48,49]. Furthermore, their intrinsic mobility limit is  $2 \times 10^5 \text{cm}^2/\text{Vs}$  [48], exceeding the mobility of semiconducting CNTs ( $\sim 1 \times 10^5 \text{cm}^2/\text{Vs}$  [50]).

The importance of GNR edge states was predicted by physics first-principles calculations [51,52]. The edges of graphene can either be zigzag or armchair, depending on the orientation of the graphene lattice edge. A recent experiment used scanning tunneling microscopy to verify this prediction, confirming that the crystallographic orientation of the edges significantly influences the electronic properties of nanometer-sized graphene [9]. By measuring the band gap of graphene samples and noting their edge chirality, they observed that nanoribbons with predominantly zigzag edges are metallic, while those with predominantly armchair edges are semiconducting. Figure 5.2 illustrates the armchair and zigzag edge states and their corresponding chirality.

GNRFETs with traditional metal source and drain electrodes have been proven experimentally and been manufactured at wafer scales [53]. Research groups have also started investigating a new class of GNRFETs that are patterned entirely from a single piece of graphene [54–56]. The idea of all-graphene design is based on metal-semiconductor junctions formed by a change in GNR chirality. GNR chirality changes occur at the bends of a graphene nanoribbon. Band gap is strongly dependent on edge orientation, so a change in the chirality from predominantly zigzag (metallic) to pre-

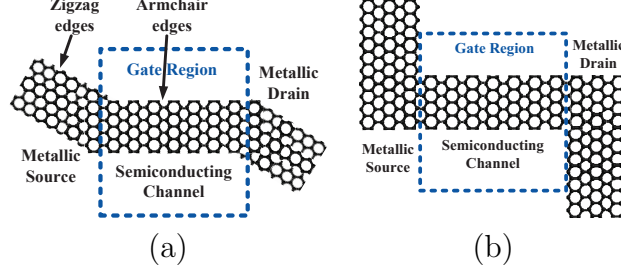


Figure 5.3: All-graphene GNRFET design: the metal-semiconductor junction is formed by the change of GNR chirality.

dominantly armchair (semiconducting) will form a metal to semiconducting junction. Figure 5.3 shows the design of GNRFETs without metal source and drain. In order to build a circuit using such transistors, metallic interconnections are needed between devices. Due to the special features of metallic GNRs, routing between the transistors becomes an interesting and unique problem. In the next section, we will show how we formulate the GNR routing problem.

### 5.3 GNR Routing Problem

Let us assume that the overall graphene sheet is oriented such that some bonds are horizontal, as shown in Figure 5.4 (other orientations of the sheet can be discussed in the same way). To obtain a semiconducting GNR, which can be abstracted as a thin rectangle with width  $< 10\text{nm}$ , we need the two sides of the rectangle to have armchair edge states. This requires the two sides of the rectangle to be oriented at 0, 60, or 120 degrees, which means that the rectangle itself should also be oriented at 0, 60, or 120 degrees (see Figure 5.4 (a)). Similarly, metallic GNRs can be considered as rectangles oriented at 30, 90, or 150 degrees (see Figure 5.4 (b)).

As a result, we can abstract the physical layout of a GNR cell as a collection of semiconducting rectangles oriented at 0, 60 and 120 degrees connected by GNR wires oriented at 30, 90 and 150 degrees. The GNR routing problem is this: Given the location and orientation (one of 0, 60 and 120 degrees) of the semiconducting terminals (a terminal is one end of the semiconducting rectangle), how do we determine the routing path of the metallic GNR wires between them so that the delays of the paths are minimized?

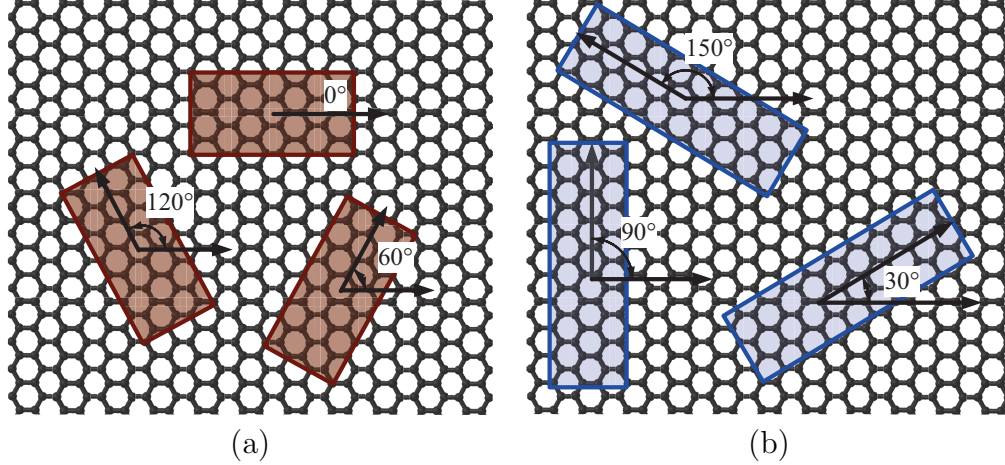


Figure 5.4: GNR orientations: (a) semiconducting, (b) metallic.

For GNR wires, two major factors contribute to the delay: *length* and *bending*. According to [57], the interconnect resistance is a constant when the length is smaller than the mean free path length, and then it goes up according to the extra length beyond the mean free path length. The mean free path length is regarded as  $1\mu\text{m}$  in [57]. This means that for terminals that are close to each other, the length contribution to the overall resistance is very small while for two pins that are far apart, the length has a more significant contribution to the resistance (and therefore to the delay).

Another major contributor to the resistance is the bending of the GNRs. Bends in a GNR cause reflection, backscattering and resonant tunnel effects [55] which then contribute to the overall resistance. Bends of different angles result in different resistance. According to the simulation done by Areshkin and White [58], a  $120^\circ$  bend introduces a resistance 3x larger than that introduced by a  $60^\circ$  bending. This is somewhat counter-intuitive because we would expect sharper bending to introduce more serious reflection. The reason behind the interesting simulation result is that the current per unit energy for the  $60^\circ$  bending exhibits a pronounced circular pattern in the energy ranges. Such a circular pattern exhibits a resonant behavior [58], which yields a current per unit energy in the “resonant cavity” of substantially higher magnitude. As a result, the 60 degree bend only shows a 12% conductance degradation compared to the straight line (no bending). On the other hand, a  $120^\circ$  bend does not exhibit the resonant behavior, and thus produces a greater reflection effect with considerably higher resistance.

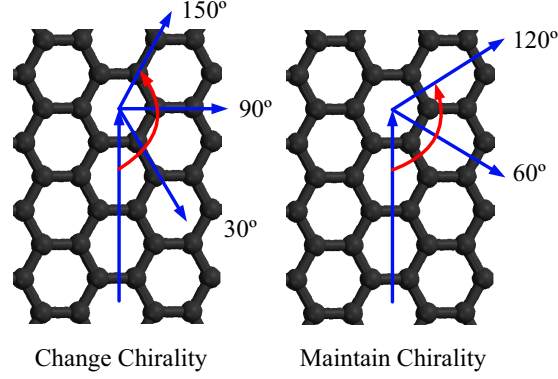


Figure 5.5: Effect of different bends on GNR chirality.

Notice that [58]’s result is based on simulation; actual resistance caused by bending still needs to be verified by future experiments.

The bending costs of other angles ( $30^\circ$ ,  $90^\circ$ , and  $150^\circ$ ) involve more complicated mechanisms because such angles imply a chirality change from the metallic GNR wires. Such changes of chirality form metal-to-semiconductor junctions and the resistance at these junctions is not reported by any literature. See Figure 5.5 for an example. Bending at  $30^\circ$ ,  $90^\circ$ , and  $150^\circ$  implies chirality changes and the formation of junctions, while at  $0^\circ$ ,  $60^\circ$  and  $120^\circ$  bending, the chirality remains the same.

Because of the above properties of GNR wires, we propose to formulate the GNR routing problem as a *minimum hybrid-cost path problem* on a triangular mesh (hybrid means the cost is a hybrid of the length cost and bending cost). Since the metallic GNRs are oriented at 30, 90, or 150 degrees, the underlying routing grid also has routing tracks along these degrees. Furthermore, we assume a uniform wiring pitch for the metallic GNRs (which is essentially the sum of the width and the separation). As a result, we will have a triangular mesh, shown in Figure 5.6, as our underlying routing grid.

We use a hybrid cost model to model the routing cost of a GNR wire. The total cost of one GNR wire is the sum of two parts: *length cost* and *bending cost*. The length cost is the length of the wire multiplied by a weight  $w_L$ . If the distance between the two terminals on the mesh is within the mean free path ( $1\mu\text{m}$  according to [57]), then  $w_L$  is a very small constant. However, if the distance exceeds the mean free path,  $w_L$  becomes larger. On the other hand, we assign different costs to different degrees of bending. We denote the cost of  $150^\circ$ ,  $120^\circ$ ,  $90^\circ$ ,  $60^\circ$ ,  $30^\circ$  bending as  $\alpha_{150^\circ}$ ,  $\alpha_{120^\circ}$ ,  $\alpha_{90^\circ}$ ,  $\alpha_{60^\circ}$ ,  $\alpha_{30^\circ}$  respec-

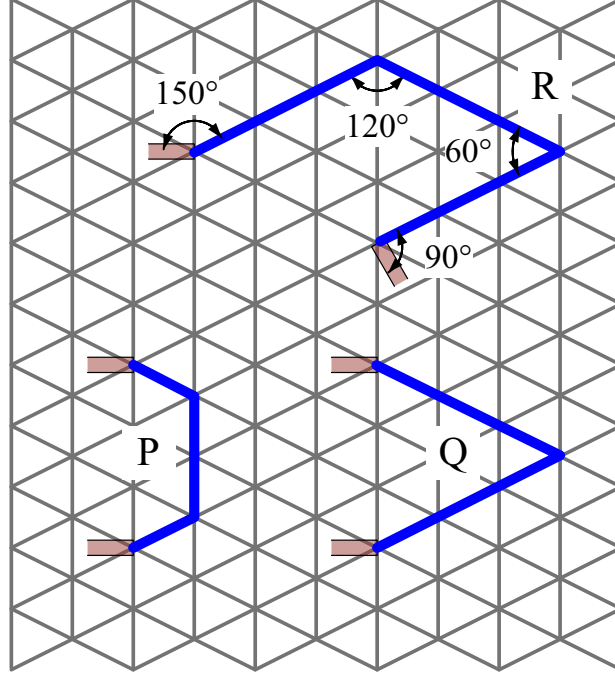


Figure 5.6: Triangular routing grid for GNR routing path. The cost of a path depends on the length and bending.

tively. From the simulations in [58], we know  $\alpha_{120^\circ} \approx 3\alpha_{60^\circ}$ . Other  $\alpha$  values are unknown for now. The total bending cost is the sum of the corresponding  $\alpha$  values over all bends along the path. One thing to notice is that since the orientations of the devices and the interconnections are different, inevitably we have to introduce bending at the two terminals of one net. The total cost of a path is the sum of the length cost and the bending cost. Let us take a look at the routing path  $R$  in Figure 5.6 as an example. At one end, the metallic GNR forms a  $150^\circ$  bending with the semiconductive GNR. At the other end, the metallic GNR forms a  $90^\circ$  bending with another semiconductive GNR. The metallic GNR has two bendings itself, one is  $120^\circ$  and the other is  $60^\circ$ . Finally, the length of the GNR is 9. Therefore, the total cost of path  $T$  is

$$C_R = 9w_L + \alpha_{150^\circ} + \alpha_{90^\circ} + \alpha_{120^\circ} + \alpha_{60^\circ} \quad (5.1)$$

The GNR routing problem is to find the path between two given terminals on a triangular mesh such that the hybrid cost is minimized. This *minimum hybrid-cost path (MHCP)* problem is very different from the traditional IC routing problem, which is a minimum edge-cost path problem on rectangular grid. Let us look at the two paths  $P$  and  $Q$  in Figure 5.6. If we use the tradi-

tional minimum edge-cost formulation, we would prefer  $P$  over  $Q$ . However,  $Q$  is more favorable in our case even though it is longer. The reason is that when the routing length is shorter than the mean free path, the contribution of length to the resistance is constant. So both paths would have the same length cost. However,  $P$  has two  $120^\circ$  bends while  $Q$  has only one  $60^\circ$  bend. Notice that  $\alpha_{120^\circ} \approx 3\alpha_{60^\circ}$ , which means  $P$ 's bending cost is much larger than  $Q$ 's bending cost. As a result,  $Q$  would be more favorable than  $P$ . This example indicates that the MHCP problem is unique and that the standard routing algorithm, maze routing (or Dijkstra's algorithm), is not applicable to the problem. In the next section, we will show how we solve this problem optimally.

## 5.4 Solving the MHCP Problem

Suppose we are given a triangular mesh  $M$  and two terminals  $S$  and  $T$ . We can construct a routing graph  $G$  if we regard every intersection of six line segments as a node and the short segments between nodes as edges in the mesh  $M$ . Each terminal corresponds to a node in  $G$ , and a path in  $G$  between two nodes is a routing path between the corresponding terminals. By assigning a cost  $w_L$  to each edge, we can capture the length cost (recall that  $w_L$  is small when the distance between the two terminals is shorter than the mean free path length and large when the distance exceeds that). However, this model does not capture the bending cost, as assigning costs to edges or nodes cannot reflect the bendings.

### 5.4.1 Graph Expansion

To consider the bending cost, we expand each node in  $G$  into six nodes (see Figure 5.7). The six edges incident to the original node in  $G$  are now incident to the six nodes respectively. We then add edges between the six nodes. The cost of such an edge depends on the angle of bending from one node to the other. If there is no bending, then the cost of the edge is 0. Table 5.1 concludes the costs of all the edges in Figure 5.7.

If a node is a terminal of a GNR-FET, then we add an extra node in the center in addition to the node expansion (see Figure 5.8). The center node

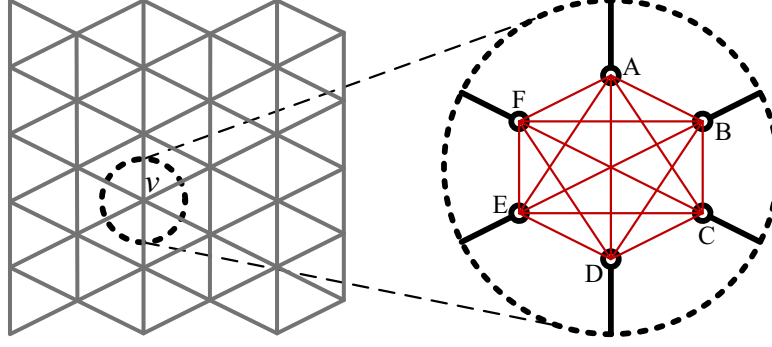


Figure 5.7: Modeling the bending cost at a node by expanding the node into six nodes, adding edges between the six nodes and assigning proper costs to the added edges.

Table 5.1: Edge costs of the graph in Figure 5.7

Edge	Cost	Edge	Cost	Edge	Cost
AB	$\alpha_{60^\circ}$	AC	$\alpha_{120^\circ}$	AD	0
AE	$\alpha_{120^\circ}$	AF	$\alpha_{60^\circ}$	BC	$\alpha_{60^\circ}$
BD	$\alpha_{120^\circ}$	BE	0	BF	$\alpha_{120^\circ}$
CD	$\alpha_{60^\circ}$	CE	$\alpha_{120^\circ}$	CF	0
DE	$\alpha_{60^\circ}$	DF	$\alpha_{120^\circ}$	EF	$\alpha_{60^\circ}$

now becomes the terminal node. Instead of connecting the six nodes, we connect the center node to the six nodes. The cost of each edge corresponds to the bending cost from the semiconducting GNR to the metallic GNR at the terminal. Table 5.2 concludes the costs of all the edges in Figure 5.8.

The edges that correspond to the original mesh segments in  $M$  are called *mesh edges*, and the edges that span between the six expanded nodes or between the terminal node and the six expanded nodes (edges in Table 5.1 and Table 5.2) are called *expanded edges*.

By performing the above graph expansion, we obtain an expanded graph  $G'$ .

**Lemma 4.** *Any path in  $M$  corresponds to a path in  $G'$  of the same cost.*

*Proof.* For any path  $P \in M$ , let us assume it consists of  $n$  edges  $\{e_1, e_2, \dots, e_n\}$ .

Table 5.2: Edge costs of the graph in Figure 5.8

Edge	Cost	Edge	Cost	Edge	Cost
TA	$\alpha_{90^\circ}$	TB	$\alpha_{150^\circ}$	TC	$\alpha_{150^\circ}$
TD	$\alpha_{90^\circ}$	TE	$\alpha_{30^\circ}$	TF	$\alpha_{30^\circ}$

These  $n$  edges should also appear in the expanded graph  $G'$  as mesh edges. By our construction in Figure 5.7, two edges  $e_i, e_{i+1}$  incident at the same node in  $M$  are now connected by an expanded edge in  $G'$ . Also,  $e_1$  and  $e_n$  are connected to the two terminal nodes through expanded edges in  $G'$ . These expanded edges and the original mesh edges form a path  $P'$  in  $G'$ . From Table 5.1 and Table 5.2, we can see that the cost of the expanded edges corresponds to the bending cost. The length cost on the mesh edges, on the other hand, remains the same when we expand the graph. Therefore, the cost of  $P'$  is equal to the hybrid cost of the path  $P$ .  $\square$

Figure 5.9 illustrates two paths in  $M$  and their corresponding paths in  $G'$ . The hybrid-costs of the paths in  $M$  are equal to the total edge-costs of the corresponding paths in  $G'$ .

#### 5.4.2 Shortest Red-Black Path Problem

Unfortunately, the reverse is not true. Not every path in  $G'$  has a corresponding path in  $M$  with the same cost. Let us look at Figure 5.10. The solid path  $AC$  corresponds to the  $120^\circ$  bending in  $M$ . Moreover, the cost of edge  $AC$  is exactly  $\alpha_{120^\circ}$ , which is consistent with the bending angle. However, the dashed path  $AFC$  does not have such consistency. While the actual bending angle is  $120^\circ$  in the mesh, the total edge cost of  $AFC$  is  $c_{AF} + c_{FC} = \alpha_{60^\circ} + 0 = \alpha_{60^\circ}$ , which is much smaller than  $\alpha_{120^\circ}$  in our case. In fact, we cannot find any corresponding path in the mesh  $M$  that has the same cost as  $AFC$ . What is even worse is that if we directly apply the

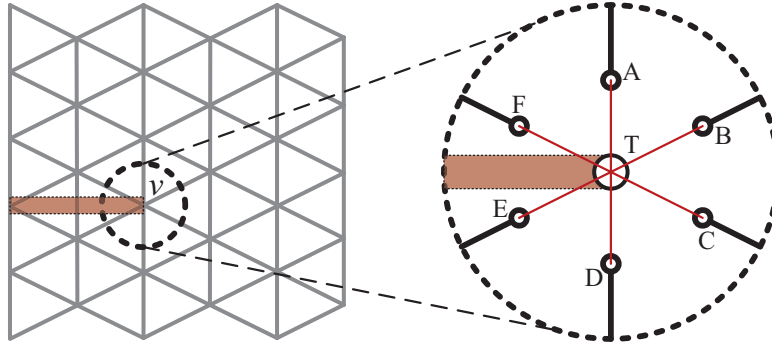


Figure 5.8: For a terminal, we add a node in the center in addition to the node expansion. Instead of connecting the six nodes, we connect the center node to the six nodes using edges of proper costs.



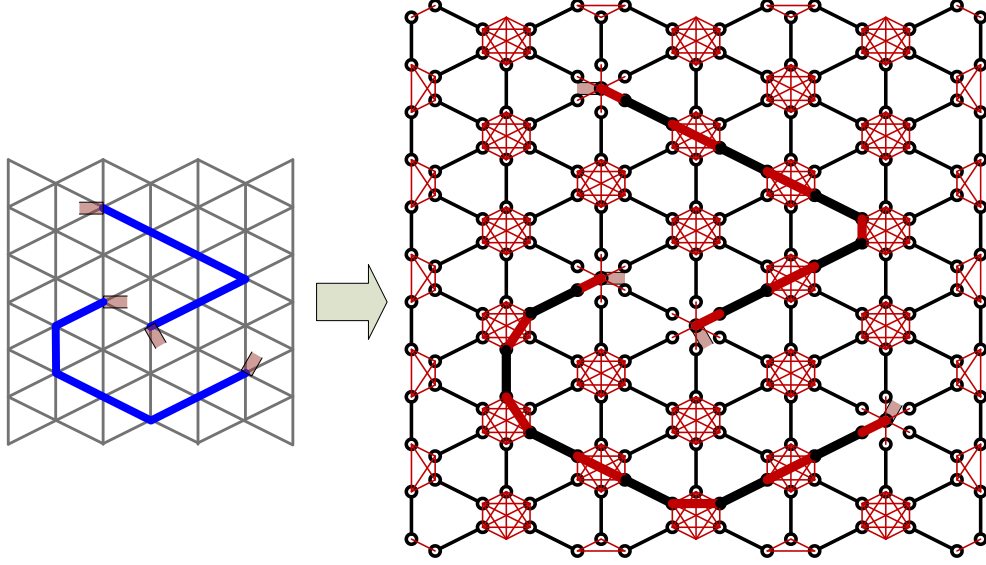


Figure 5.9: The routing on the mesh  $M$  (left) and the corresponding path in the expanded graph  $G'$  (right).

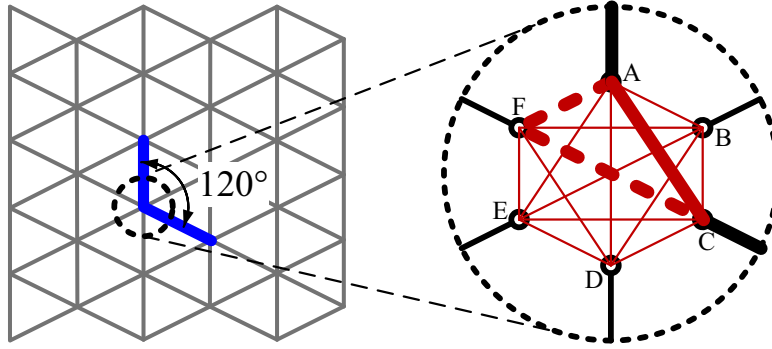


Figure 5.10: Even though the actual route in  $M$  (left) has a bending of  $120^\circ$ , directly applying the shortest path algorithm will result in a route (dashed route) with bending cost  $\alpha_{60^\circ}$  in  $G'$ .

shortest path algorithm on  $G'$ , the algorithm will choose route  $AFC$  over  $AC$  because  $AFC$  has smaller cost. Eventually, the result produced by the algorithm would not correspond to any actual routing path in  $M$  with correct hybrid-cost.

Before we explain how we address the above issue, we would like to introduce some definitions and denotations. We color the edges in  $G'$  by two colors: the mesh edges are colored **black** (they are also drawn as thick edges in Figure 5.7 and Figure 5.8) and the expanded edges are colored **red** (they are also drawn as thin edges in Figure 5.7 and Figure 5.8). With the defini-

tions above, we can define the cost of an edge  $e$  in  $G'$  more clearly:

$$\begin{cases} c_e = w_L & \text{if } e \text{ is black} \\ c_e \text{ defined by Table 5.1 or Table 5.2} & \text{if } e \text{ is red} \end{cases} \quad (5.2)$$

A graph is called a *red-black graph* if all its edges are either red or black. A path in a red-black graph is called a *red-black path* if the edges along the path have alternating colors.

**Lemma 5.** *A path in  $G'$  between two terminal nodes has a corresponding path in  $M$  iff the path is a red-black path.*

*Proof.* If a path  $P'$  is a red-black path in  $G'$ , let us consider the path  $P$  on the mesh  $M$  formed by the edges corresponding to the black edges of path  $P'$ . It is easy to check that each red edge in  $P'$  correctly models the cost of the corresponding bending in  $P$  in all circumstances (our graph models in Figure 5.7 and Figure 5.8 are constructed for this purpose), thus  $P$  is the corresponding path of  $P'$  in  $M$ . On the other hand, if a path  $P'$  in  $G'$  is not a red-black path, it must contain at least two consecutive red edges. This is because no two black edges share the same node in the construction of  $G'$ . Since  $P'$  contains at least two consecutive red edges and the red edge cost correspond to the bending cost, the path is counting at least two bending costs at the same node. As a result, the cost of  $P'$  does not correspond to the hybrid cost of any path in  $M$ .  $\square$

Notice that both paths in Figure 5.9 are red-black paths. Therefore, they both have corresponding paths in the mesh with consistent cost. In order to find the minimum hybrid-cost path in the mesh, we need to solve the following *shortest red-black path problem*:

**Problem 1.** *Given a red-black graph  $G'$ , find a red-black path between two specified nodes with the minimum total edge cost along the path.*

Directly applying Dijkstra's algorithm would not solve this problem. In [47], we proposed a modified Dijkstra's algorithm to solve it. The idea was to keep a color label at each node in addition to the traditional distance label. The color label keeps the color of the edge that updates the minimum distance from source to the current node so that when we expand from current node, we would avoid use the same color edge. Unfortunately, the algorithm

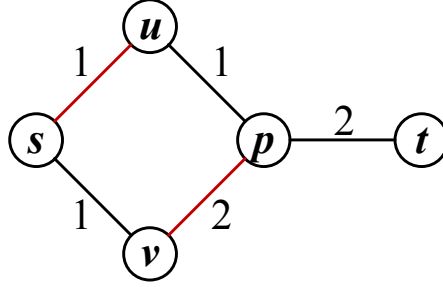


Figure 5.11: An example on which the modified Dijkstra's algorithm cannot produce the correct result.

is flawed. It may not produce the shortest path in certain situations. Figure 5.11 illustrate this issue. If we start our modified Dijkstra's algorithm at  $s$ , we will eventually update the distance label of  $p$  to 2 and color label of  $p$  to black. Then, since edge  $(p, t)$  is also black, we cannot expand from  $p$  to  $t$ . As a result, the algorithm would claim there exist no red-black path from  $s$  to  $t$  while the path  $\{s, v, p, t\}$  is one.

In this chapter, we propose to expand the graph again and use the classical Dijkstra's algorithm to solve this problem.

### 5.4.3 Second Graph Expansion

Given an undirected red-black graph  $G'$ , we construct a directed graph  $G''$  as follows:

- For each node  $v \in G'$ , we construct two nodes: a black node  $v^b$  and a red node  $v^r$ .
- For each undirected edge  $(u, v) \in G'$ , we construct two directed edges that have the same cost as  $(u, v)$ :
  - Black edges  $(u^r, v^b)$  and  $(v^r, u^b)$ , if  $(u, v)$  is black.
  - Red edges  $(u^b, v^r)$  and  $(v^b, u^r)$ , if  $(u, v)$  is red.

The construction of nodes and edges in  $G''$  is illustrated in Figure 5.12. The expanded graph of our routing models in Figure 5.7 and Figure 5.8 are shown in Figure 5.13 and Figure 5.14, respectively.

From the construction of  $G''$ , we can see that  $G''$  has the following property:

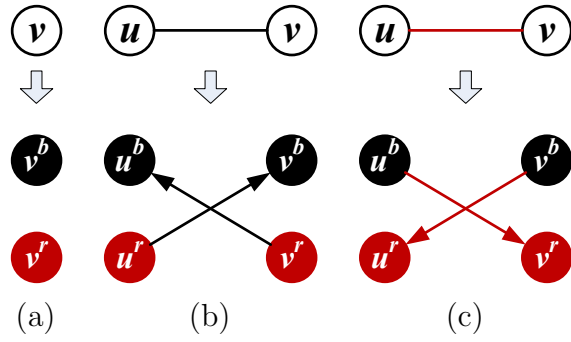


Figure 5.12: Construction of nodes and edges in the second graph expansion: (a) expand a node; (b) expand a black edge; (c) expand a red edge.

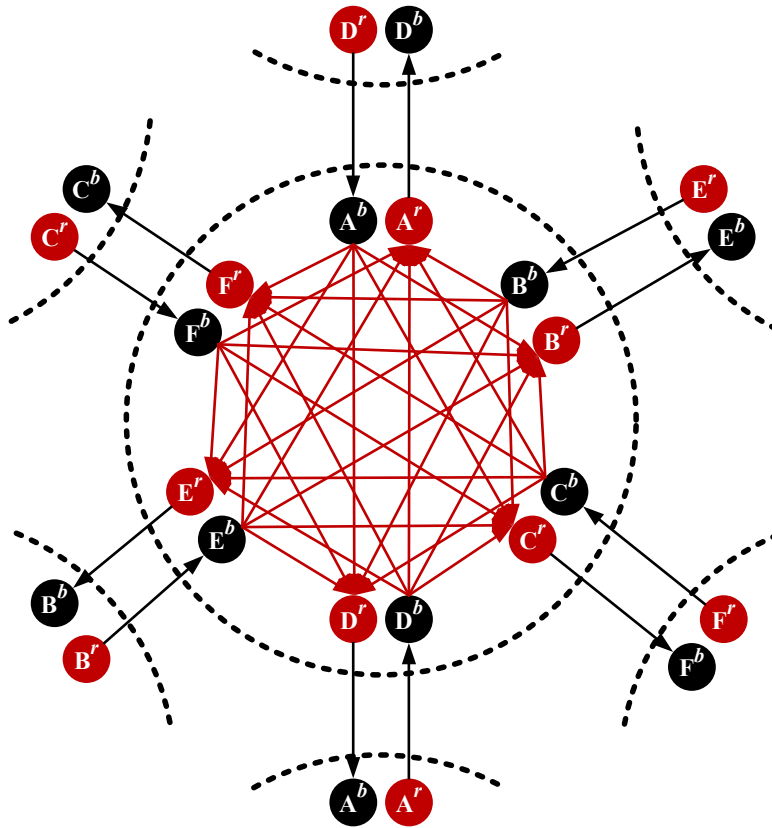


Figure 5.13: Expanded graph of Figure 5.7.

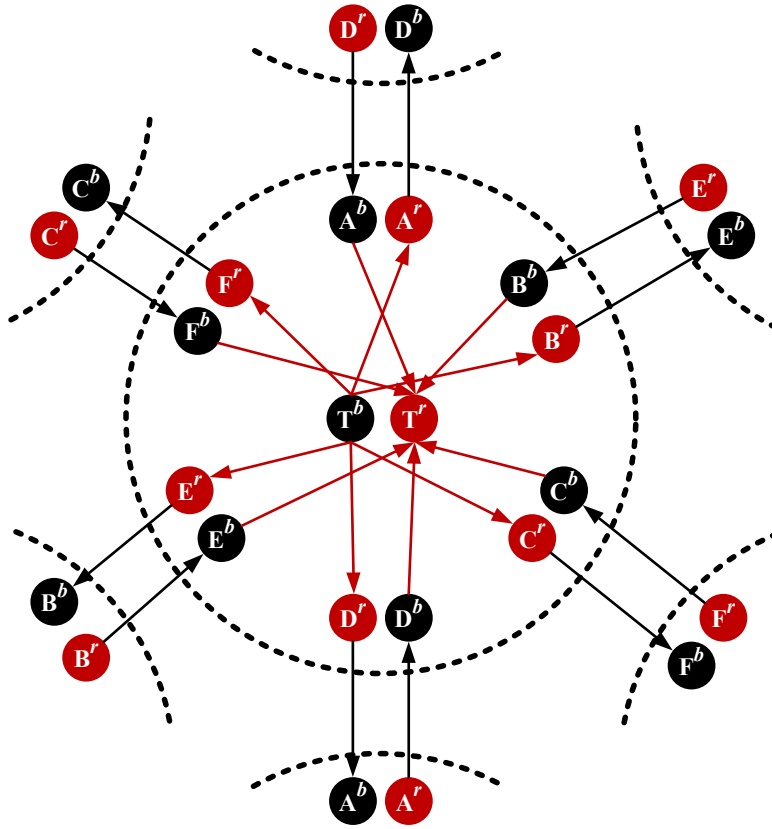


Figure 5.14: Expanded graph of Figure 5.8.

**Remark 1.** For a directed edge  $(u, v)$  in  $G''$ , node  $v$  has the same color as the edge, while node  $u$  has the different color than the edge.

Therefore, we have the following lemma:

**Lemma 6.** Any path in  $G''$  is a red-black path.

*Proof.* Given any path  $P$  in  $G''$ , assume that  $(u, v)$  and  $(v, w)$  are two consecutive edges in  $P$ . According to our observation, node  $v$  has the same color as edge  $(u, v)$ , and a different color than edge  $(v, w)$ , which indicates that  $(u, v)$  and  $(v, w)$  are of different colors. This means that any two consecutive edges in path  $P$  are of different colors, which completes the proof.  $\square$

This leads to the following theorem:

**Theorem 8.** There exists a one-to-two correspondence between red-black paths in  $G'$  and directed paths in  $G''$ . Furthermore, the mapping is color and cost preserving.

*Proof.* Given any red-black path  $P = \{v_1, v_2, \dots, v_n\} \in G'$ , we can turn this path into a directed path by assuming a direction on it. There are only two possible directions: from  $v_1$  to  $v_n$  and from  $v_n$  to  $v_1$ . Therefore, the path can be turned into two directed paths  $D_1 = \{v_1, v_2, \dots, v_n\}$  and  $D_2 = \{v_n, v_{n-1}, \dots, v_1\}$  in  $G'$ . Each of the two directed paths corresponds to a directed path in  $G''$ . The correspondence is as follows: each edge  $(v_i, v_j)$  in  $D_1$  or  $D_2$  correspond to an edge  $(v_i^{\bar{c}}, v_j^c)$  in  $G''$  in which  $c$  is the color of edge  $(v_i, v_j)$  (either  $r$  or  $b$ ) and  $\bar{c}$  is the opposite of  $c$ . Such corresponding edges in  $G''$  form two paths.

For example, if edge  $(v_1, v_2)$  is black and  $(v_{n-1}, v_n)$  is red in  $P$ . Then the following two paths in  $G''$  correspond to  $P$ :

- $\{v_1^r, v_2^b, v_3^r, \dots, v_{n-1}^b, v_n^r\}$
- $\{v_n^b, v_{n-1}^r, v_{n-2}^b, \dots, v_2^r, v_1^b\}$

Now we need to show that the corresponding edges in  $G''$  indeed form continuous paths. Given any two consecutive edges  $(v_i, v_{i+1})$  and  $(v_{i+1}, v_{i+2})$  in  $P$ , let us assume the color of  $(v_i, v_{i+1})$  is  $c$ . Then the color of  $(v_{i+1}, v_{i+2})$  must be  $\bar{c}$  because  $P$  is a red-black path. The corresponding edges in  $G''$  are  $(v_i^{\bar{c}}, v_{i+1}^c)$  and  $(v_{i+1}^c, v_{i+2}^{\bar{c}})$ . This means the two edges are connected at

$v_{i+1}^c$ ). The same thing happens for  $D_2$ . Therefore, the edges in  $G''$  forms two connected paths, one correspond to  $D_1$  and the other correspond to  $D_2$ .

Given any path  $\{v_1^x, v_2^x, \dots, v_n^x\} \in G''$  (the superscript  $x$  means that we do not care about the color of the node), we know the path is a red-black path according to Lemma 6. By removing the color tag of the nodes, we obtain a path  $\{v_1, v_2, \dots, v_n\} \in G'$ . Since our edge construction preserves the color,  $\{v_1, v_2, \dots, v_n\}$  is also a red-black path.

The cost and color preservation of the mapping is because that our construction of the edge in  $G''$  preserves the color and cost of the edges in  $G'$ .  $\square$

In order to obtain the shortest red-black path between two nodes  $s$  and  $t$  in  $G'$ , we can compute four shortest paths in  $G''$ :

- Shortest path from  $s^b$  to  $t^b$ ;
- Shortest path from  $s^b$  to  $t^r$ ;
- Shortest path from  $s^r$  to  $t^b$ ;
- Shortest path from  $s^r$  to  $t^r$ ;

The shortest of the four shortest paths corresponds to the shortest red-black path in  $G'$ . In fact, we could simplify the procedure by introducing a super source  $S$  and a super sink  $T$  in  $G''$  and add zero cost directed edges from  $S$  to  $s^r$  and  $s^b$  and from  $t^r, t^b$  to  $T$ . Then the shortest path from  $S$  to  $T$  corresponds to the shortest red-black path in  $G'$ . Theorem 8 guarantees the correctness of this algorithm.

We have discussed how to obtain the shortest red-black path in an undirected graph, similar technique can be used to find the shortest red-black path in directed graph. The only difference is that we only construct one directed edge in  $G''$  for each directed edge in  $G'$  and the direction of the constructed edge is consistent with the original edge in  $G'$ . Fig 5.15 illustrates the idea.

The advantage of our graph expansion approach is that it does not require any modification of the shortest path algorithm. After the graph expansion, one can use any existing shortest path algorithm/software on the expanded graph. Then simple transformation can be applied to obtain the minimum hybrid-cost path in the original graph.

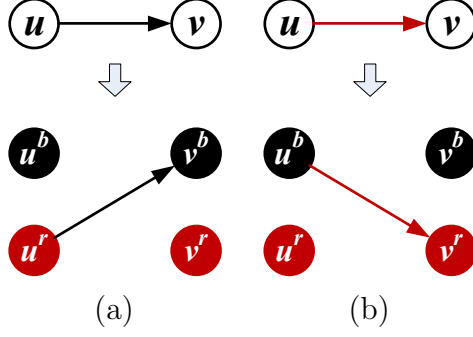


Figure 5.15: Construction of nodes and edges for directed graph: (a) expand a black edge; (b) expand a red edge.

## 5.5 Negotiated Congestion Routing Scheme

The algorithm presented in the previous section routes only a single net. In practical designs, we need to route multiple nets together. In this section, we describe the negotiated congestion based routing scheme we use to route multiple nets.

The negotiated congestion based routing scheme was first proposed for FPGA routing [7] and then was adopted by various IC global routers [4–6]. Recently, Ma et al. [38] proposed to use this routing scheme to solve a planar routing problem and the experimental results look very promising. Because graphene has a planar structure, the GNR routing must also be planar. Therefore, we use the routing scheme in [38] to route multiple GNR wires.

In this routing scheme, routability is achieved by forcing all of the nets to negotiate for a resource and thereby determine which net needs the resource most. Some nets may use shared resources that are in high demand if all alternative routes utilize resources in even higher demand. Other nets will tend to spread out and use resources in lower demand. All of the nets are iteratively rerouted until no more resources are shared.

In the negotiated congestion routing scheme, the actual cost  $\gamma_e$  of an edge  $e \in G'$  is a combination of the hybrid-cost of the edge and the congestion cost of the edge:

$$\gamma_e = c_e + h_v \times p_v \quad (5.3)$$

In the equation,  $c_e$  is the edge cost defined by Eq. (5.2).  $h_v$  and  $p_v$  describes the congestion cost. They are defined only on red edges (they are 0 on black



Table 5.3: Experimental Results

Test Case	Grid Size	#Net	Routability	Wire Length	#Bending		Runtime (s)
	#R×#C				60°	120°	
<i>ex1</i>	40×30	10	100%	718	19	1	15.91
<i>ex2</i>	50×40	20	100%	1755	35	10	450.26
<i>ex3</i>	50×40	22	100%	1972	34	11	170.67
<i>ex4</i>	50×50	30	100%	2169	49	3	162.52
<i>ex5</i>	50×50	40	100%	3776	76	19	402.85
<i>ex6</i>	60×60	50	100%	4488	98	32	677.18

edges). Notice that all the red edges are expanded from the nodes in the mesh. In Figure 5.7 and Figure 5.8, the red edges all correspond to one node  $v$  in the mesh.  $h_v$  and  $p_v$  are defined on that node and all the red edges that correspond to node  $v$  share the same  $h_v$  and  $p_v$  cost.  $h_v$  is the history cost of  $v$ , and  $p_v$  denotes the number of nets currently passing  $v$ . Our router works as follows. Before the routing starts,  $h_v$  and  $p_v$  are set to zero for all mesh nodes. In the initial iteration, all of the nets are routed one by one using the shortest red-black path algorithm. Rip-up and reroute will then be performed if congestion exists, i.e., if a mesh node is occupied by more than one net. In each iteration of rip-up and reroute, every net is rerouted, even if the net does not pass through a congested area. In this way, nets passing through uncongested areas can be diverted to make room for other nets in congested regions. At the beginning of each rip-up and reroute iteration, the history cost  $h_v$  of every congested node  $v$  is incremented by  $\Delta$ , which is a user-defined parameter. The procedure terminates when all of the routes are disjoint.

## 5.6 Experimental Result

Based on the algorithm and the routing scheme presented in the previous sections, we implemented a router for GNR routing in C++. Since no data exist for GNR circuits, we built six test cases and used them to test our router. The experiments were performed on a Linux system with a 2.0GHz CPU and 2GB memory. The results are shown in Table 5.3. We can observe that we have many more 60° bendings than 120° bendings. This is because the cost of a 120° bending is 3 times the cost of a 60° bending in our settings

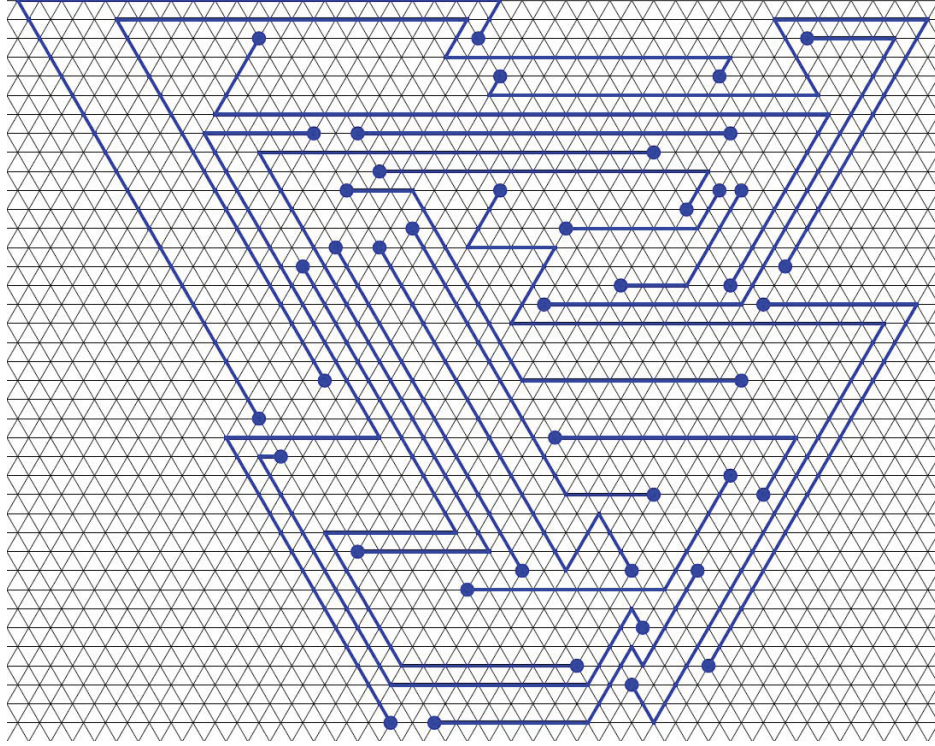


Figure 5.16: The resultant routing of *ex3*.

(we choose 3x according to the simulation results in [58]). Figure 5.16, which illustrates the results of *ex3*, also shows that  $60^\circ$  bending is more favorable than  $120^\circ$  bending. We can see that some routes would rather use longer wire length and  $60^\circ$  bendings to avoid  $120^\circ$  turns. These observations indicate that our router is capable of handling the hybrid cost of length and bendings.

## 5.7 Conclusion

This is the first work that studies the graphene nanoribbon (GNR) routing problem. We formulate GNR routing as a minimum hybrid-cost shortest path problem. The underlying routing grid is a triangular mesh due to the properties of metallic GNRs. In order to model this hybrid-cost problem, we apply graph expansion on the triangular mesh and then introduce a shortest red-black path problem. We also propose an algorithm that optimally solves the shortest red-black path problem and therefore optimally solves the minimum hybrid-cost shortest path problem. We fit our algorithm into a negotiated congestion based routing scheme and test it by experiment. The

experimental results show that our routing algorithm effectively solves the hybrid-cost routing problem.

# CHAPTER 6

## THE ROUTING RELIABILITY PROBLEM

### 6.1 Introduction

One of the most important tasks in physical design of VLSI circuits is routing. The quality and reliability of routing have great impact on the performance of the chip. There are numerous algorithmic studies in literature on the routing problem, discussing how to connect different modules with shortest possible wiring while trying to avoid congestion [4], [7], [5], [6]. In this chapter, however, we focus on the reliability aspect of routing.

Conventional CMOS devices face an increasing number of challenges as the feature sizes scale down, including increased wire resistivity due to surface scattering and grain-boundary scattering, increased leakage power, significant mobility degradation, and large dopant fluctuations [59]. In the meantime, chemically synthesized nanoscale materials, such as nanowires, carbon nanotubes (CNT), and graphene nanoribbons (GNR), have been shown to have favorable device properties, new device characteristics, and large integration capability through new fabrication techniques. These nano-scale devices have significant potential to revolutionize the fabrication and integration of electronic systems and operate beyond the perceived scaling limitations of traditional CMOS [8]. Recently, top-gated GNR field-effect transistors of various gate lengths have been fabricated with peak cutoff frequencies up to 26GHz for a 150nm gate length [46]. These nanomaterials are also found to possess metallic properties and thus can be used as interconnection wiring [9]. Yan *et al.* studied the GNR routing problem in [47], and a negotiated congestion based router is developed. However, in practice, the GNR wire segments have a connection defective rate. Some GNR wire segments will be more vulnerable to wear-out effects, and some wire segments may fail to connect during manufacturing [9]. To be more exact,

each wire segment has a survival probability, and thus has a chance to fail. In such a circumstance, an interconnection between two terminals realized by a simple path, like in traditional VLSI routing, is vulnerable to defect, since this interconnection fails as long as any wire segment in this path fails. This definitely creates more challenges for routing reliability, and inspires us to develop new routing strategies.

In this chapter, we study the routing reliability problem, and propose an algorithm flow to solve it. In this routing reliability problem, each edge in the routing graph is associated with a cost (wire length, congestion, etc.), as well as a survival probability. A simple path between two terminals, like in traditional routing, is liable to defect. Our idea is to reinforce the reliability of the original routing path by adding redundant wiring segments in such a way that the survival probability of the interconnection is maximized with a reasonable overhead of routing resources. Kahng *et al.* studied a similar non-tree routing problem in [60] for reliability and yield improvement, where they tried to augment the edge connectivity of a routed net by adding redundancies. Their approach is limited to regular grid graphs with uniform survival probability distribution. Moreover, the edge connectivity of a routing does not directly reflect its survival probability. Our proposed approach works on general graphs with arbitrary probability distributions, and it is briefly described as follows. Given a routing graph with an  $s$ - $t$  routing path (assume this path is accomplished by a traditional router), we first use min-cost max-flow algorithm to identify a set of good candidate wiring segments with low cost and high survival probability. A dynamic programming procedure is then employed to optimally select the segments among the candidate set to add to the routing. The proposed approach is tested on both general graphs and grid graphs, and the results are promising. The survival probability of the  $s$ - $t$  connection can be increased from 67.93% to 94.17% on average, with a reasonable overhead of routing resources. The results also show that our proposed approach is very efficient, even large test cases can be accomplished within one second. This makes it possible to integrate our algorithm flow into industrial routers to address the reliability issue.

The rest of this chapter is organized as follows: Section 6.2 gives some preliminaries on the background knowledge. Section 6.3 defines the routing reliability problem; Section 6.4 presents our algorithm flow for solving this problem; Section 6.5 reports the experimental results, and Section 6.6

concludes this chapter. This work is published in [61].

## 6.2 Preliminaries

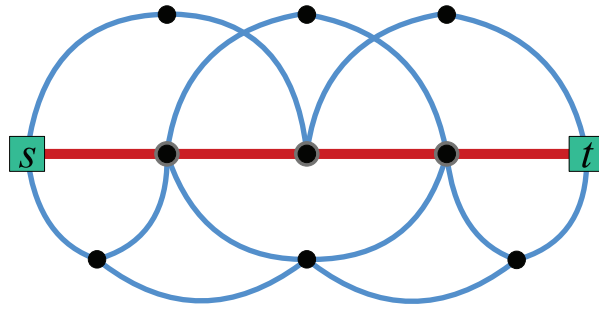
In traditional VLSI routing, an interconnection between two terminals in a routing graph is implemented by a path connecting them, as the path connecting  $s$  and  $t$  in the routing graph shown in Figure 6.1(a). However, in case of high defective rate, each edge in the routing graph fails independently with a probability; consequently, the connection between  $s$  and  $t$  has a probability to fail. We want to evaluate the reliability of the  $s$ - $t$  connection by *s-t connection survival probability*, which is defined as follows.

**Definition 3.  $s$ - $t$  Connection Survival Probability** - *Given an undirected graph  $G(V, E)$  where each edge  $e \in E$  survives with a probability  $Pr(e)$ , and thus to fail with a probability  $1 - Pr(e)$ , the connection between two terminals  $s$  and  $t$  ( $s, t \in V$ ) contains a set of edges  $P \subseteq E$ ; the  $s$ - $t$  connection survival probability is the probability that  $s$  and  $t$  get connected by the edges in  $P$ .*

Let us assume, w.l.o.g., that each edge  $e$  in the routing graph of Figure 6.1 survives with a probability  $Pr(e) = 0.95$  (note that in practice the survival probabilities of the edges can be different), and let  $Pr(s, t)$  denote the  $s$ - $t$  connection survival probability. Thus, it is easy to compute in Figure 6.1(a) that  $Pr(s, t) = (0.95)^4 = 81.45\%$ . In our routing reliability problem, an original path connecting  $s$  and  $t$  is given, and we want to enhance the reliability of the  $s$ - $t$  connection by adding redundancy segments. A redundancy segment is defined as follows.

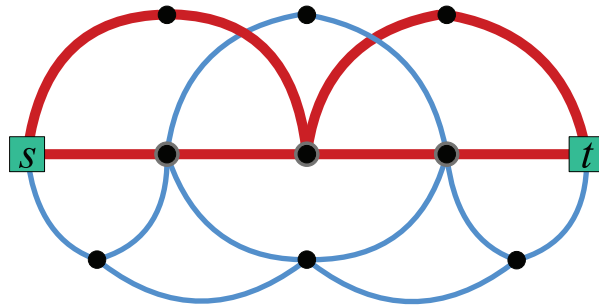
**Definition 4. Redundancy Segment** - *A path  $g$  with its two endpoints as  $u$  and  $v$  is a redundancy segment if (1)  $u$  and  $v$  are lying on the original  $s$ - $t$  path, (2)  $u$  and  $v$  are the only two vertices of  $g$  lying on the original  $s$ - $t$  path, and (3)  $g$  does not share any edges with the original  $s$ - $t$  path.*

In Figure 6.1(b), two redundancy segments are added into the  $s$ - $t$  connection, and  $Pr(s, t)$  is augmented to 99.05%, which is calculated as  $(1 - (1 - 0.95^2)^2)^2$ . The computation becomes more complicated, but it is still easy to figure out. More redundancy is added in Figure 6.1(c), and  $Pr(s, t)$  immediately becomes nontrivial to compute.



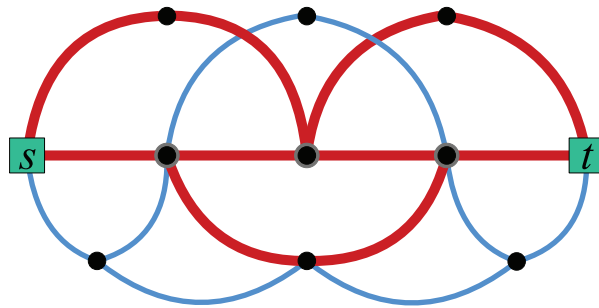
$$Pr(s, t) = 81.45\%$$

(a)



$$Pr(s, t) = 99.05\%$$

(b)



$$Pr(s, t) = ?$$

(c)

Figure 6.1: (a) The original  $s$ - $t$  connection is a path (the path with thick edges),  $Pr(s, t) = 81.45\%$ ; (b) Two redundancy segments are added into the  $s$ - $t$  connection,  $Pr(s, t) = 99.05\%$ ; (c) One more redundancy segment is added, and  $Pr(s, t)$  is nontrivial to compute.

The computation of the  $s$ - $t$  connection survival probability turns out to be a classic problem in reliability theory. It is called the two-terminal reliability problem in the communication community, and it was shown to be  $\#P$ -complete [62] [63]. That is, it is in a complexity class at least as intractable as NP-complete and therefore seems unlikely to have polynomial time solutions. This indicates that the problem we are trying to solve is theoretically hard, since we are not even able to accurately evaluate the reliability of a candidate solution, whose edges contained in the  $s$ - $t$  connection have been determined. As far as we know, our whole problem of selecting a set of redundancy segments with the budget constraint while trying to maximize the  $s$ - $t$  reliability has never been addressed in the literature.

We observe that the  $s$ - $t$  connection survival probability can be efficiently computed if the  $s$ - $t$  connection forms a two-terminal series-parallel graph, with  $s$  and  $t$  as the two terminals.

**Definition 5. Two-Terminal Series-Parallel Graph-** *A graph with two pre-assigned terminals  $s$  (source) and  $t$  (sink) is a two-terminal series-parallel graph, if it can be constructed by a sequence of series compositions and parallel compositions starting from a set of single-edge graphs [64]. Given two graphs  $X$  and  $Y$  as input, both of which are two-terminal graphs, the two operations are described as follows:*

- *Series composition: Merge the sink of  $X$  and the source of  $Y$ ; let the source of  $X$  and the sink of  $Y$  be the source and sink of the resultant graph, respectively.*
- *Parallel composition: Merge the source of  $X$  and the source of  $Y$  to be the source of the resultant graph; merge the sink of  $X$  and the sink of  $Y$  to be the sink of the resultant graph.*

The series composition and parallel composition are illustrated in Figure 6.2. Based on the definition, it is easy to see that the  $s$ - $t$  connection in Figure 6.1(b) forms a two-terminal series-parallel graph.

In the literature, series-parallel graphs are used to approximate the two-terminal reliability problem on general graphs, e.g., in [65], Aboelfotoh *et al.* efficiently bounded the two-terminal reliability problem via approximation by series-parallel graphs. Also, in our routing reliability problem, a series-parallel graph is sufficient to model the  $s$ - $t$  connection. This is the fact



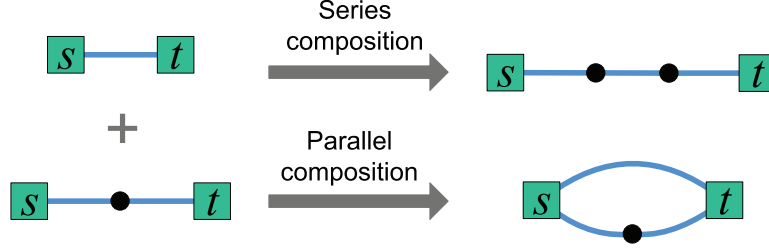


Figure 6.2: Series composition and parallel composition.

that the  $s$ - $t$  connection cannot be too complicated in practice, as the limited routing resources would not allow too much redundancy added (the cost of which is another important criterion of our routing reliability problem). Therefore, in our approach, we are going to model the  $s$ - $t$  connection as a two-terminal series-parallel graph; in this way, the complexity of the problem we are solving is brought down with only a marginal sacrifice on the solution quality.

### 6.3 Problem Definition

In this chapter, we study the routing reliability problem, where we aim to maximally reinforce the reliability of a given  $s$ - $t$  routing path by adding in redundancy segments, such that the total cost of the redundancy segments does not exceed a given budget.

**Definition 6. Routing Reliability Problem** - *Given an undirected routing graph  $G(V, E)$  where each edge  $e \in E$  is associated with a survival probability  $Pr(e)$  and a cost  $c(e)$ , as well as an original routing path  $P$  between two terminals  $s$  and  $t$  ( $s, t \in V$ ), the objective is to maximize the  $s$ - $t$  connection survival probability by adding a set of redundancy segments to  $P$  in such a way that the total cost of the redundancy does not exceed a given budget  $B$ .*

### 6.4 Algorithm Flow for the Routing Reliability Problem

Our algorithm flow can be divided into two stages:

1. A set  $S$  of redundancy segments is generated from the routing graph via the min-cost max-flow algorithm, such that each redundancy segment contains edges with high survival probability and low cost. In other words, the redundancy segments in  $S$  are good choices to add to the  $s$ - $t$  connection  $P$ .
2. The redundancy segments in  $S$  are optimally selected into the  $s$ - $t$  connection  $P$  by dynamic programming. The resultant  $s$ - $t$  connection  $P$  is a two-terminal series-parallel graph.

#### 6.4.1 Generating Candidate Redundancy Segments Via Min-cost Max-Flow

We adopt the min-cost max-flow algorithm to simultaneously generate a set  $S$  of good candidate redundancy segments. The two terminals  $s$  and  $t$  are set to be the source and sink of our flow network, respectively. The edges in the routing graph are properly oriented to make our flow network a directed graph. We are trying to push max-flow from  $s$  to  $t$ . Max-flow enables us to generate as many candidate redundancy segments as possible, so that the dynamic programming procedure in the next stage will have a wider range of choices for optimal selection. The cost of each edge in the flow network needs to reflect both its survival probability and its intrinsic cost. Intuitively, the cost of an edge  $e$  in the flow network should be smaller if its survival probability  $Pr(e)$  is larger and/or its intrinsic cost  $c(e)$  is smaller, so that the min-cost flow will prefer to travel in the edges with large  $Pr(e)$  and small  $c(e)$ , and consequently identify the good candidate redundancy segments. The candidate redundancy segments generated by the min-cost max-flow are later feed to the dynamic programming procedure as input for optimal selection. The dynamic programming procedure requires that the candidate redundancy segments in  $S$  are vertex-disjoint, since otherwise it is not guaranteed that the resultant  $s$ - $t$  connection is a series-parallel graph. Note that here “vertex-disjoint” refers to the vertices that are not lying on the original  $s$ - $t$  path, thus two candidate redundancy segments are allowed to share the vertices lying on the original  $s$ - $t$  path, i.e., they can have the same endpoints. This “vertex-disjoint” requirement is enforced by splitting a vertex  $v$  into two vertices  $v_{in}$  and  $v_{out}$  which are connected by a directed

edge with capacity set to 1.

In the following, we describe in detail how our flow network  $G'(V', E')$  is constructed based on the routing graph  $G(V, E)$ :

- **Source and sink:** The two terminals  $s$  and  $t$  are set to be the source and sink of our flow network, respectively. The supply of  $s$  and the demand of  $t$  are both set to  $|V|$ . This ensures that the max-flow is  $|V|$ , which is enough as there are at most  $|V|$  vertex-disjoint redundancy segments.
- **Edge orientation and vertex splitting:** The flow network  $G'(V', E')$  needs to be a directed graph, while the routing graph  $G(V, E)$  is an undirected graph. Thus we need to orient the edges in  $G(V, E)$  in order to transform it into a flow network. Let us refer to the edges in the original  $s$ - $t$  path as trunk edges, and refer to the other edges as branch edges. For each branch edge  $e$  with  $u$  and  $v$  as its two endpoints, we make another copy of  $e$  and orient the two copies in opposite directions, i.e., one copy from  $u$  to  $v$  and the other from  $v$  to  $u$ . This is due to the fact that the flow in the network should be able to travel from  $u$  to  $v$  as well as from  $v$  to  $u$ . For each trunk edge, we simply orient it towards  $t$  (the original  $s$ - $t$  path becomes a directed path from  $s$  to  $t$ ). Let us denote the set of oriented trunk edges by  $E_1$ , and denote the set of oriented branch edges by  $E_2$ . In addition, each vertex  $v$  not lying on the original  $s$ - $t$  path is split into an input vertex  $v_{in}$  and an output vertex  $v_{out}$ , i.e., the incoming edges of  $v$  are now directed to  $v_{in}$ , and the outgoing edges of  $v$  are now emanating from  $v_{out}$ . A directed edge from  $v_{in}$  to  $v_{out}$  is added. Let  $E_3$  denote this set of newly added edges from  $v_{in}$  to  $v_{out}$  for each vertex  $v$  not lying on the original  $s$ - $t$  path. Thus  $E' = E_1 \cup E_2 \cup E_3$ . The vertex splitting technique is used to enforce that the redundancy segments found are vertex-disjoint, by setting the capacity of the edge from  $v_{in}$  to  $v_{out}$  as 1 (the detailed capacity setting is discussed later). Note that the vertices lying on the original  $s$ - $t$  path do not need to be split as the redundancy segments can have the same endpoints. Figure 6.3 shows the flow network constructed from the routing graph in Figure 6.1(a).
- **Edge capacity setting:** Let  $cap(e)$  denote the capacity of an edge

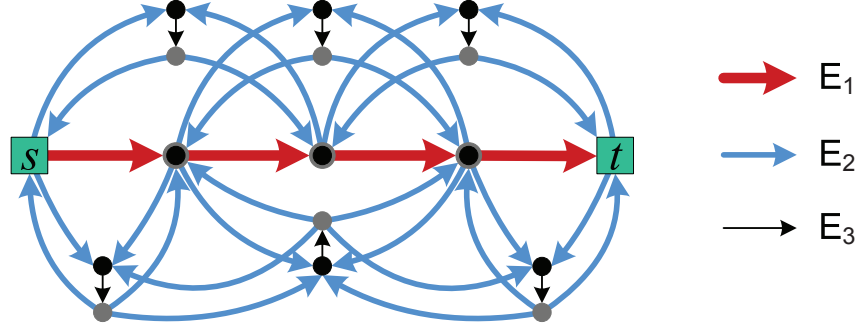


Figure 6.3: Illustration of the flow network construction.

$e \in E'$ . For each edge  $e \in E_2 \cup E_3$ ,  $\text{cap}(e)$  is set to 1 so that the redundant segments generated by flow are guaranteed to be vertex-disjoint. For each edge  $e \in E_1$ ,  $\text{cap}(e)$  is set to  $|V|$ , which ensures that  $|V|$  units of flow (our max-flow) can be sent from source to sink.

- **Edge cost setting:** Let  $c'(e)$  denote the cost of an edge  $e \in E'$  in the flow network. For each edge  $e \in E_1$ ,  $c'(e)$  is set to a large constant  $M$ . In this way, the trunk edges are expensive for the flow to travel along, so the flow in the trunk will escape to the branches whenever there is a chance, and thus identify the good candidate redundancy segments along the way. In our implementation,  $M$  is set to be the sum of the cost of all the edges in  $E_2$ . For each edge  $e \in E_2$ ,  $c'(e)$  has to be able to reflect both the survival probability  $\text{Pr}(e)$  and the cost  $c(e)$ . We use  $(-\log \text{Pr}(e))$  to reflect the survival probability portion (the reason is explained later), so we set  $c'(e) = \lambda c(e) + (1 - \lambda)(-\log \text{Pr}(e))$ , where  $0 < \lambda < 1$ .  $\lambda$  is a user defined parameter and can be tuned to adjust the relative importance between  $\text{Pr}(e)$  and  $c(e)$ . In our experiments, our approach is tested with different  $\lambda$  values (see Section 6.5 for details). For each edge  $e \in E_3$ ,  $c'(e)$  is simply set to 0.

We now explain our cost setting of the edges in  $E_2$ . It is actually guided by the following simpler problem. Suppose, in the extreme case, each edge  $e \in E$  is only associated with a survival probability  $\text{Pr}(e)$  ( $c(e) = 0$ ), and we are asked to find a path between  $s$  and  $t$  with highest survival probability. That is, we want to find a path  $P$  between  $s$  and  $t$  such that  $\prod_{e \in P} \text{Pr}(e)$  is maximized. It is not difficult to figure out that this problem reduces to finding the shortest path between  $s$  and  $t$  if we set the length of

each edge  $e$  as  $(-\log Pr(e))$ , since  $\prod_{e \in P} Pr(e)$  is maximized if and only if  $\sum_{e \in P} (-\log Pr(e))$  is minimized. Note that the correctness of this reduction also lies in the fact that the edge length  $(-\log Pr(e))$  is a nonnegative number, as  $0 < Pr(e) \leq 1$ . That is, the cost of an edge  $e$  should be set to  $(-\log Pr(e))$  if  $c(e) = 0$ . Whenever  $c(e)$  is a positive number, as in our case, it is very natural and reasonable to set the edge cost  $c'(e)$  as a weighted sum of  $c(e)$  and  $(-\log Pr(e))$ .

After solving the min-cost max-flow on the network  $G'(V', E')$  constructed, a set of vertex-disjoint candidate redundancy segments can be obtained. The branch edges with nonzero flow value automatically indicate the candidate redundancy segments. The min-cost max-flow problem can be solved by the capacity scaling based algorithm in  $O(|E|^2 \log |V| \log U)$  time [66], where  $U$  is the upper bound on the edge capacities. In our case, the time complexity is  $O(|E|^2 \log^2 |V|)$  as  $U = |V|$ .

#### 6.4.2 Optimal Series-Parallel $s$ - $t$ Connection Generation by Dynamic Programming

After the set  $S$  of candidate redundancy segments is obtained, we need to make a selection out of  $S$  to add to the  $s$ - $t$  connection  $P$ , since the cost budget  $B$  may be exceeded if we add all of them to  $P$ . As we have mentioned, we aim to generate a series-parallel  $s$ - $t$  connection, whose survival probability can be easily computed. In this section, we are going to present a dynamic programming procedure. This procedure generates an optimal series-parallel  $s$ - $t$  connection by optimally selecting redundancy segments out of  $S$ , in the sense that the  $s$ - $t$  connection survival probability  $Pr(s, t)$  is maximized while the total cost of the redundancy segments selected does not exceed the budget  $B$ .

In order to facilitate the presentation of our dynamic programming procedure, we first label the vertices along the original  $s$ - $t$  path with  $1, 2, \dots, n$ , with  $s$  labeled with 1 and  $t$  labeled with  $n$  (assume there are  $n$  vertices on this path). We denote the vertex labeled with  $i$  by  $v_i$ ,  $\forall i \in \{1, 2, \dots, n\}$ . Note that here we are using the routing graph  $G(V, E)$ , not the flow network  $G(V', E')$ . For every two vertices  $v_i$  and  $v_j$  on the original  $s$ - $t$  path ( $i < j$ ), there may be several candidate redundancy segments connecting them in  $S$ ,

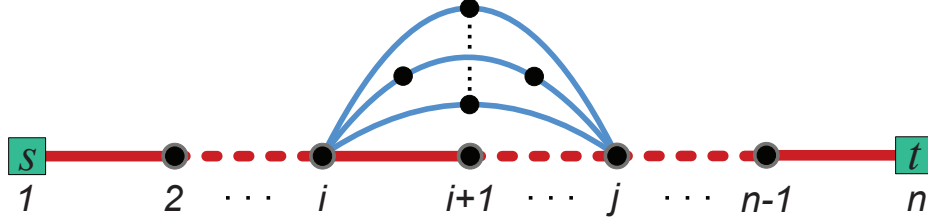


Figure 6.4: Labeling of the vertices on the original  $s$ - $t$  path; Multiple redundancy segments connecting  $v_i$  and  $v_j$  may exist.

as shown in Figure 6.4. For the ease of explanation, we first assume that, for any two vertices  $v_i$  and  $v_j$ , there is at most one candidate redundancy segment in  $S$  connecting them. Our dynamic programming procedure based on this assumption is presented in Section 6.4.2. Later we will show in Section 6.4.2 that our dynamic programming procedure can be easily extended to handle general case where multiple redundancy segments connecting  $v_i$  and  $v_j$  may exist.

### The Special Case

In the following, we use  $g_{i,j}$  to denote the redundancy segment connecting  $v_i$  and  $v_j$ , if it exists. Let  $c(g_{i,j})$  denote the cost of  $g_{i,j}$ , which is computed as the sum of the cost of the edges belonging to  $g_{i,j}$ , so  $c(g_{i,j}) = \sum_{e \in g_{i,j}} c(e)$ . It is important to notice that, in our dynamic programming procedure, the edge costs are assumed to be integers. This is true in practice, as the length and congestion cost of the edges in a routing graph are integers in most routing implementations. Therefore, the cost of the redundancy segments are also integers. Let  $Pr(g_{i,j})$  denote the survival probability of  $g_{i,j}$ , which is computed as the product of the survival probabilities of the edges belonging to  $g_{i,j}$ , so  $Pr(g_{i,j}) = \prod_{e \in g_{i,j}} Pr(e)$ .

Our dynamic programming procedure is developed based on the series-parallel structure of the resultant  $s$ - $t$  connection. We observe that the optimal  $s$ - $t$  connection comes from either a parallel composition or a series composition of two smaller series-parallel graphs. We use  $Pr(1, n, B, 1)$  to denote the survival probability of the optimal  $s$ - $t$  connection, where the first and second parameter means that we are finding the optimal connection from  $s$  to  $t$  ( $s$  and  $t$  are labeled with 1 and  $n$  respectively), the third parameter indicates that our total cost budget is  $B$ , and the last parameter “1” implies that we

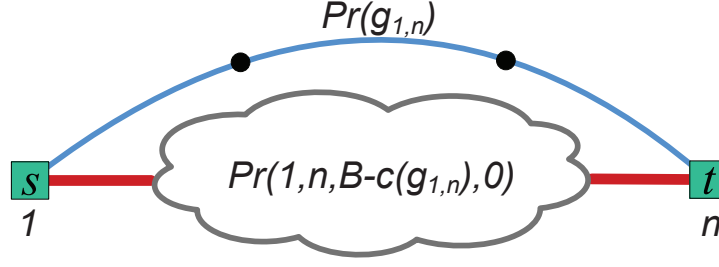


Figure 6.5: The optimal solution contains  $g_{1,n}$ .

are allowed to use redundancy segment  $g_{1,n}$  connecting  $s$  and  $t$ . These four parameters determine the size of the problem we are solving. By guessing if the redundancy segment  $g_{1,n}$  is selected into the optimal solution or not, the problem decomposes into two subproblems of smaller size:

1.  **$g_{1,n}$  is selected into the optimal solution:** In this case, the optimal solution is the parallel composition of  $g_{1,n}$  and the optimal solution of a subproblem  $P_1$ , where  $P_1$  is to use the remaining budget  $(B - c(g_{1,n}))$  and the remaining redundancy segments to find an optimal  $s$ - $t$  connection, as illustrated in Figure 6.5. The survival probability of the optimal solution of  $P_1$  can be denoted by  $Pr(1, n, B - c(g_{1,n}), 0)$ , where the third parameter indicates that our budget is now  $B - c(g_{1,n})$ , and the last parameter “0” implies that we are no longer allowed to use  $g_{1,n}$ . Therefore, we have

$$Pr(1, n, B, 1) = Pr(1, n, B - c(g_{1,n}), 0) \parallel Pr(g_{1,n}).$$

Note that here, for the ease of presentation, we use “ $\parallel$ ” to denote a parallel composition.  $Pr(a) \parallel Pr(b)$  means the survival probability of the series-parallel graph obtained by a parallel composition of two subgraphs with survival probabilities as  $Pr(a)$  and  $Pr(b)$ , respectively, thus  $Pr(a) \parallel Pr(b) = 1 - (1 - Pr(a))(1 - Pr(b))$ .

2.  **$g_{1,n}$  is not selected into the optimal solution:** In this case, the problem reduces to a subproblem  $P_2$ , which is to use the budget  $B$  to find an optimal  $s$ - $t$  connection without using  $g_{1,n}$ . As a result, we have

$$Pr(1, n, B, 1) = Pr(1, n, B, 0).$$

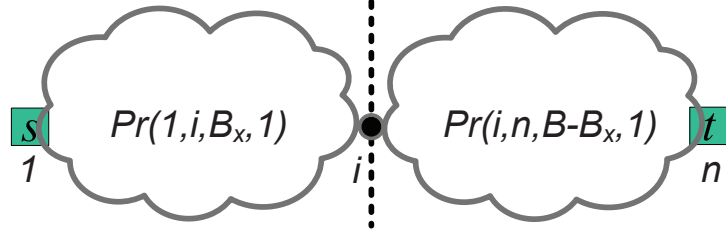


Figure 6.6: The optimal solution does not contain  $g_{1,n}$ .

We now figure out how to compute  $Pr(1, n, B, 0)$ . Observe that, in this case, the optimal  $s$ - $t$  connection comes from a series composition, as illustrated in Figure 6.6. We can enumerate all the possibilities by considering the vertex at which the series composition occurs. Suppose the series composition occurs at  $v_i$  ( $1 < i < n$ ), we can separately consider the subproblem from  $s$  ( $v_1$ ) to  $v_i$  and the subproblem from  $v_i$  to  $t$  ( $v_n$ ). Note that we also need to enumerate all possible budget distributions on the two subproblems, e.g., if the cost budget for the subproblem from  $s$  to  $v_i$  is  $B_x$ , the cost budget for the subproblem from  $v_i$  to  $t$  would be  $B - B_x$ . Therefore, by enumerating  $v_i$  and  $B_x$ , we have

$$Pr(1, n, B, 0) = \max_{1 < i < n, 0 \leq B_x \leq B} \{Pr(1, i, B_x, 1) \cdot Pr(i, n, B - B_x, 1)\}.$$

Note that we only enumerate all the possible integer values of  $B_x$ , since all the edges costs are assumed to be integers.

To summarize, we have the following recursions:

$$Pr(1, n, B, 1) = \max \left\{ \begin{array}{l} Pr(1, n, B - c(g_{1,n}), 0) \parallel Pr(g_{1,n}), \\ Pr(1, n, B, 0) \end{array} \right\} \quad (6.1)$$

$$Pr(1, n, B, 0) = \max_{1 < i < n, 0 \leq B_x \leq B} \{Pr(1, i, B_x, 1) \cdot Pr(i, n, B - B_x, 1)\} \quad (6.2)$$

By recursively computing the solutions of all the subproblems, i.e., the  $Pr(i, j, B_x, 0)$ s and  $Pr(i, j, B_x, 1)$ s ( $1 \leq i < j \leq n, 0 \leq B_x \leq B$ ), we can get optimal  $s$ - $t$  connection together with its survival probability  $Pr(1, n, B, 1)$ . Note that a memoization technique needs to be applied along the way to store the solutions of the subproblems, which is a standard dynamic programming technique. According to equation (1), each  $Pr(i, j, B_x, 1)$  can be computed in constant time from the solutions of its subproblems; while according to



equation (2), the computation of each  $Pr(i, j, B_x, 0)$  takes  $O(nB)$  time. There are  $O(n^2B)$  subproblems to compute, thus it follows that the time complexity of our dynamic programming procedure is  $O(n^3B^2)$ .

### The General Case

Let us now consider the general case where multiple redundancy segments connecting  $v_i$  and  $v_j$  may exist in  $S$ . We show our dynamic programming procedure can be easily extended to take care of this variation. Let  $k_{i,j}$  denote the number of candidate redundancy segments connecting  $v_i$  and  $v_j$ . We index these redundancy segments from 1 to  $k_{i,j}$ , and denote the one indexed with  $l$  ( $1 \leq l \leq k_{i,j}$ ) as  $g_{i,j,l}$ . Similarly, we denote the cost and survival probability of  $g_{i,j,l}$  by  $c(g_{i,j,l})$  and  $Pr(g_{i,j,l})$ , respectively.

We use  $Pr(1, n, B, k_{1,n})$  to represent the survival probability of the optimal  $s$ - $t$  connection instead of  $Pr(1, n, B, 1)$ , since there are  $k_{1,n}$  candidate redundancy segments connecting  $s$  and  $t$ . By guessing whether each of these redundancy segments is contained in the optimal solution one by one, from  $g_{1,n,k_{1,n}}$  down to  $g_{1,n,1}$ , the problem decomposes into a collection of subproblems of smaller size. We first guess if  $g_{1,n,k_{1,n}}$  is contained in the optimal solution. Similarly to the argument in the special case, we have

$$\begin{aligned} & Pr(1, n, B, k_{1,n}) \\ = & \max \left\{ \begin{array}{l} Pr(1, n, B - c(g_{1,n,k_{1,n}}), k_{1,n} - 1) \parallel Pr(g_{1,n,k_{1,n}}), \\ Pr(1, n, B, k_{1,n} - 1) \end{array} \right\} \end{aligned}$$

We continue guessing if each of the remaining ones is contained in the optimal solution, so we have

$$\begin{aligned} & Pr(1, n, B, k_{1,n} - 1) \\ = & \max \left\{ \begin{array}{l} Pr(1, n, B - c(g_{1,n,k_{1,n}-1}), k_{1,n} - 2) \parallel Pr(g_{1,n,k_{1,n}-1}), \\ Pr(1, n, B, k_{1,n} - 2) \end{array} \right\} \\ & \dots \\ & Pr(1, n, B, 1) \\ = & \max \left\{ \begin{array}{l} Pr(1, n, B - c(g_{1,n,1}), 0) \parallel Pr(g_{1,n,1}), \\ Pr(1, n, B, 0) \end{array} \right\} \end{aligned}$$

If none of these  $k_{1,n}$  redundancy segments is contained in the optimal solution, the optimal solution comes from a series composition. Similar to the argument in the special case, we have

$$Pr(1, n, B, 0) = \max_{1 < i < n, 0 \leq B_x \leq B} \{Pr(1, i, B_x, k_{1,i}) \cdot Pr(i, n, B - B_x, k_{i,n})\}$$

Similarly, the optimal solution together with its survival probability  $Pr(1, n, B, k_{1,n})$  can be obtained by recursively computing the solutions of the subproblems and using memoization. The time complexity remains to be  $O(n^3 B^2)$ . Although things get more complicated in the general case, the time complexity is still determined by the computation of subproblems with the last parameter as “0” (the  $Pr(i, j, B_x, 0)$ s), as all the other subproblems can be computed in constant time. We have the following theorem claiming the optimality of our dynamic programming procedure.

**Theorem 9.** *Given the original  $s$ - $t$  path with  $n$  vertices, a set of vertex-disjoint candidate redundancy segments, as well as a cost budget  $B$ , the optimal series-parallel  $s$ - $t$  connection can be computed by dynamic programming in  $O(n^3 B^2)$  time.*

## 6.5 Experimental Results

We implement our algorithms in C++. To justify our approach, we perform two sets of experiments, which are described in details below. All the experiments are performed on a Linux workstation with a 3.3GHz CPU and 32GB memory.

### 6.5.1 Test on General Graphs with Randomized Edge Costs and Survival Probabilities

We first test our approach on a set of general graphs, with the number of vertices ranging from 30 to 3000. The costs  $c(e)$  and the survival probabilities  $Pr(e)$  of the edges are randomly generated within a certain range, e.g.,  $Pr(e)$  is a random number between 0.8 and 1. The results are reported in Table 6.1. The two columns “ $|V|$ ” and “ $|E|$ ” indicate the number of vertices and edges in

Table 6.1: Experimental results on general graphs with randomized edge costs and survival probabilities

Test Cases	Graph Size		$n$	$Pr(s, t)$ Init (%)	$B$	$c(e)$ Avg. (x)	$-\log Pr(e)$ Avg. (y)	$\lambda$	$\lambda x : (1 - \lambda)y$	# seg. MCMF	# seg. DP	$Pr(s, t)$ DP (%)	Runtime (s)
	$V$	$E$											
Ex1	30	90	11	61.70	70	1.98	0.19	0.1613	2:1	18	11	93.15	0.01
								0.0877	1:1	18	11	93.15	0.01
								0.0459	1:2	18	11	93.15	0.01
Ex2	100	700	24	69.78	72	2.00	0.11	0.0984	2:1	68	10	97.26	0.14
								0.0517	1:1	70	12	97.44	0.13
								0.0266	1:2	72	15	97.69	0.14
Ex3	500	2000	23	62.04	100	2.08	0.12	0.0955	2:1	50	6	91.14	0.22
								0.0502	1:1	49	6	90.39	0.23
								0.0257	1:2	51	7	90.83	0.22
Ex4	1000	10000	23	80.88	100	1.99	0.11	0.0968	2:1	125	9	98.47	0.31
								0.0509	1:1	124	8	97.67	0.30
								0.0261	1:2	127	7	98.04	0.33
Ex5	3000	15000	27	65.26	150	2.00	0.10	0.0969	2:1	104	7	89.37	0.98
								0.0509	1:1	104	6	93.05	0.94
								0.0261	1:2	104	5	91.80	0.96
Avg.	-	-	-	67.93	-	2.01	0.126	-	-	-	-	94.17	-

the routing graph, respectively. Column “ $n$ ” gives the number of vertices on the original  $s$ - $t$  path, whose initial survival probability is provided in column “ $Pr(s, t)$  Init”. In this set of experiments, we use the most reliable path  $p$  between  $s$  and  $t$  as the original  $s$ - $t$  path, i.e., the path  $p$  with minimum  $\sum_{e:e \in p} (-\log Pr(e))$ . We can see that the survival probability of the most reliable  $s$ - $t$  path is only 67.93% on average. Column “ $B$ ” shows the cost budget. The two columns “ $c(e)$  Avg.” and “ $-\log Pr(e)$  Avg.” give the average values of  $c(e)$  and  $(-\log Pr(e))$  of all the edges in the graph. For each test case, our algorithm flow is performed with three different  $\lambda$  values, so that we can see the difference when the relative importance between  $c(e)$  and  $(-\log Pr(e))$  changes while generating redundancy segments using flow. The value of  $\lambda$  is properly set to make the ratio  $\frac{\lambda Avg(c(e))}{(1-\lambda) Avg(-\log Pr(e))}$  as 2, 1 and 0.5, respectively. The two columns “# seg.MCMF” and “# seg.DP” show the number of redundancy segments selected by min-cost max-flow and dynamic programming, respectively. We can see from the table that “# seg.MCMF” is almost the same when  $\lambda$  changes, however, they are actually different segments. When  $\lambda$  is larger, a segment contains more edges with smaller  $c(e)$ ; when  $\lambda$  is smaller, a segment contains more edges with larger  $Pr(e)$ . The column “ $Pr(s, t)$  DP” shows the  $s$ - $t$  connection survival probability  $Pr(s, t)$  after dynamic programming. The results show that our proposed approach is very effective. We can see that  $Pr(s, t)$  can be boosted from 67.93% to 94.17% on average, with a reasonable amount of budget. The results also show that our proposed approach is very efficient, even large test cases can be accomplished within one second. This makes it possible to integrate our algorithm flow into industrial routers to address the reliability issue.

### 6.5.2 Test on a 3D Grid Graph with Uniform Edge Costs and Survival Probabilities

We then test our approach on a 3D grid graph of size  $20 \times 20 \times 20$  (with 8000 vertices and 22800 edges). All the edges in this graph have unit cost. Also, the survival probabilities of the all the edges are set to be the same. The given  $s$ - $t$  path contains 20 vertices. In this set of experiments, we would like to see the tradeoff between the given budget  $B$  and the resultant  $Pr(s, t)$ . Five rounds of experiments are performed on this grid graph, where the

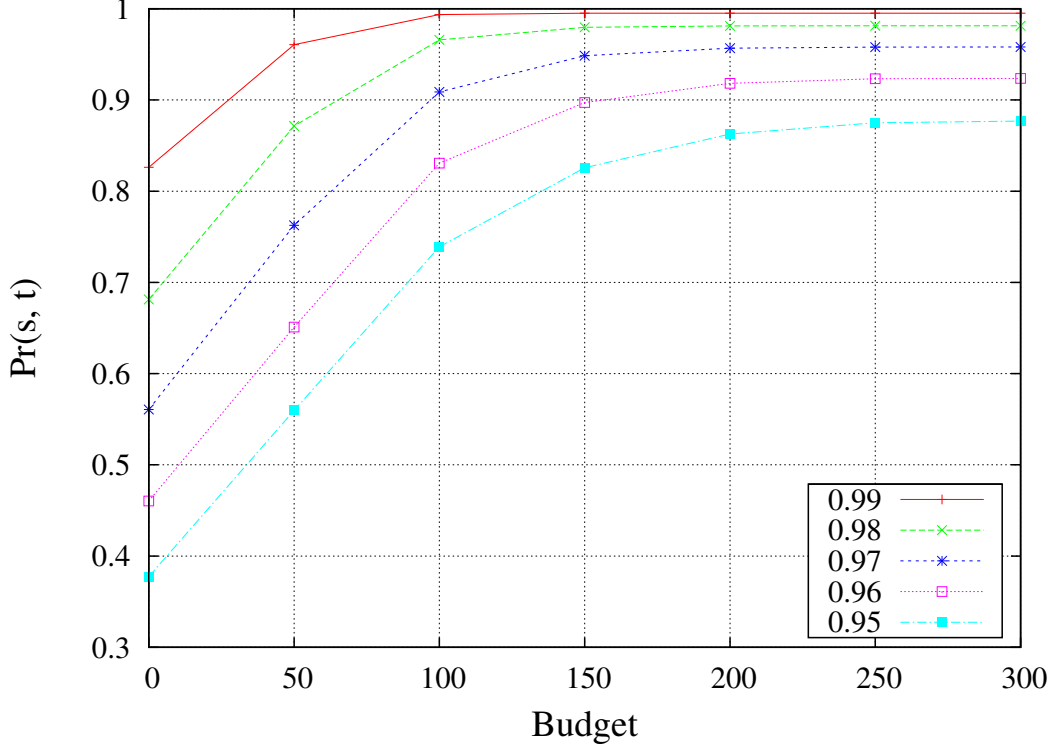


Figure 6.7: Experimental results on the 3D grid graph with uniform edge costs and survival probabilities. The tradeoff between  $Pr(s, t)$  and  $B$  is visualized with a curve for each  $Pr(e)$ .

survival probability of the edges  $Pr(e)$  is set to be 0.95, 0.96, 0.97, 0.98 and 0.99, respectively. In each round, our algorithm flow is performed six times with different budgets. Consequently, in each round, a curve of the resultant  $Pr(s, t)$  versus the budget  $B$  can be plotted, as displayed in Figure 6.7. In each curve, the value of  $Pr(s, t)$  with budget as zero show the survival probability of the original  $s-t$  path, which is relatively low.  $Pr(s, t)$  can be greatly increased by adding redundancies. We can conclude from the results that (1) when  $Pr(e)$  is lower, more budget is needed for  $Pr(s, t)$  to achieve a certain value; (2) when  $Pr(e)$  is higher, the increase of  $Pr(s, t)$  gets more marginal with the increase of budget.

## 6.6 Concluding Remarks

In this chapter, the routing reliability problem is introduced and studied. We analyze this problem and find that it is theoretically hard. We tackle this problem by dividing it into stages, and consequently propose a practical

and efficient algorithm flow. Promising experimental results validate our proposed approach.

# CHAPTER 7

## TRIPLE PATTERNING AWARE ROUTING

### 7.1 Introduction

As the minimum feature size keeps shrinking, and the availability of the next generation lithography methods (EUV, e-beam direct write, etc.) is further delayed, double patterning lithography (DPL) is commonly recognized as a feasible lithography process for 20nm technology nodes [10–13]. DPL increases pitch size and enhances resolution by decomposing the features on a critical layer into two masks. The conflicting features with spacing between them less than a predefined threshold  $d_{min}$  have to be assigned onto different masks. Whenever necessary, a feature can be further sliced to resolve conflicts, which introduce *stitches*. However, in the circumstance of high density layout, it is still possible that some conflicts cannot be resolved with stitches [67]. Moreover, the introduced stitches can lead to yield loss due to overlay error [10]. The problem of layout decomposition with conflicts and stitches minimization for DPL has been extensively studied [67,68]. Recently, Tang *et al.* have shown in [69] that this problem is polynomial time solvable and provided an optimal algorithm for it.

As technology continues to scale to 14nm node, the scalability of DPL is further challenged. The complexity of layout decomposition grows higher, and stitches become more costly as they are more sensitive to overlay error. As DPL is being pushed to its limit, triple patterning lithography (TPL) is a considerable and natural extension along the paradigm of DPL to alleviate the situation. Industry has already explored the test-chip patterns with triple patterning or even quadruple patterning [70]. Yu *et al.* studied the layout decomposition problem for TPL in [71] and formulated the problem as Integer Linear Programming. With an extra mask to accommodate the features, TPL can be used to (1) eliminate the unresolvable conflicts and

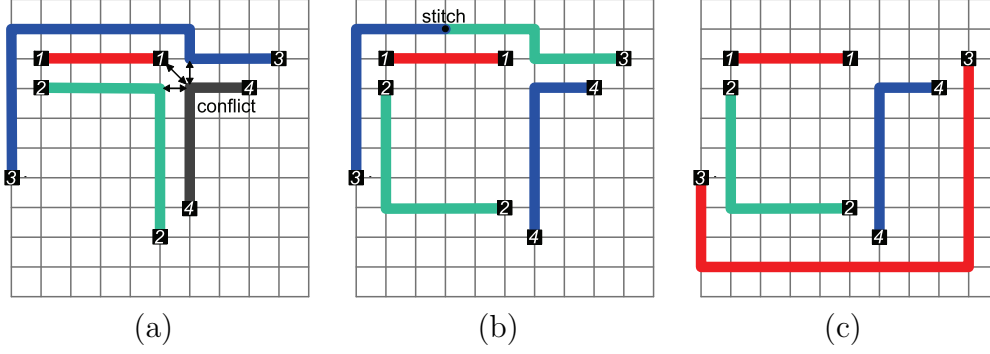


Figure 7.1: (a) The routing solution cannot be successfully decomposed due to the conflict; (b) The routing solution can be decomposed at the expense of introducing a stitch; (c) The routing solution can be decomposed without stitching.

reduce the number of stitches while maintaining the same pitch size as DPL (for 14nm node); (2) further increase the pitch size and improve the depth of focus (DOF) (for 10nm node and beyond).

A layout configuration without considering TPL/DPL at design stage can make the layout hard to decompose, and redesigning an indecomposable layout requires high ECO efforts. It is observed that most hard-to-decompose patterns originate from routing wires [72], so considering TPL/DPL during design time, especially the detailed routing stage, can significantly benefit the layout decomposition step. Figure 7.1 gives an example for illustration in the context of TPL. Figure 7.1(a) shows an indecomposable layout consisting of four nets. All four nets conflict with each other so they need to be assigned to different masks (colors), however, only three masks (colors) are available. If net 2 is implemented with an alternative route, the layout can be decomposed at the expense of introducing a stitch on net 3, as shown in Figure 7.1(b). In Figure 7.1(c), net 3 is rerouted, and the layout can be decomposed without stitching. Several previous works studied the problem of optimizing routing for double patterning. Cho *et al.* proposed the first double patterning aware detailed router in [72]. They developed a heuristic which is a modified Dijkstra's shortest path algorithm to take into account the conflicts and stitches when routing one net. Lin and Li developed a double patterning aware gridless router in [73]. They maintain a conflict graph during routing and use it to guide the routes of the incoming nets. Sun *et al.* explored in [74] post-routing layer assignment for DPL optimization. As far



as we know, no previous work has addressed triple patterning aware routing. Note that throughout this chapter, we interchangeably use the terminologies triple patterning and triple patterning lithography (TPL), as well as double patterning and double patterning lithography (DPL).

In this chapter, we study triple patterning aware routing and compare it with double patterning aware routing in 14nm technology node. We first propose a graph model that correctly models the cost of conflicts and stitches in TPL. By replacing each vertex in the routing grid with the graph model and performing shortest path algorithm on the expanded graph, the optimal path with mask/color assignment for one net can be computed, in presence of previously routed and colored nets. Our proposed graph model is a unified model that can be extended to handle multiple patterning and can also be tailored for double patterning. We then develop a negotiated congestion based scheme to resolve conflicts. The regions with conflicts are penalized and the nets are iteratively rerouted and re-colored. Experimental results show that this scheme is very effective and all conflicts can be resolved within a small number of iterations in our test cases. An effective heuristic is also developed to ensure approximately balanced utilization of the three masks. To the best of our knowledge, this is the first triple patterning aware detailed router developed in literature. We also implement a double patterning aware detailed router by adapting our graph model for double patterning, and compare it with the triple patterning version in 14nm node. Experimental results show that, using TPL, the conflicts can be resolved much more easily and the stitches can be significantly reduced in contrast to DPL.

The rest of this chapter is organized as follows: Section 7.2 states the triple patterning aware routing problem; Section 7.3 introduces our proposed graph model and describes how to route a single net on the expanded routing graph; Section 7.4 presents our negotiated congestion based scheme to resolve conflicts, as well as the heuristic to balance the routing on the three masks; Section 7.5 reports the experimental results, and Section VI concludes this chapter. This work is published in [75].

## 7.2 Problem Formulation

In TPL, we have three masks available to accommodate all the features within one layer. For the ease of discussion, we use three colors, namely, RED, GREEN and BLUE, to represent the three masks. In TPL aware detailed routing, we perform simultaneous routing and color assignment. If the spacing between two pieces of wire assigned with the same color is smaller than a predefined minimum spacing requirement  $d_{min}$ , a conflict is caused and the two pieces of wire will contribute to the total conflicting wire length. If a piece of wire changes its color assignment at some point, as net 3 in Figure 7.1(b), a stitch is introduced. We want to minimize the total conflicting wire length as well as the number of stitches. The triple patterning aware detailed routing problem is stated as follows.

**Definition 7. Triple Patterning Aware Detailed Routing** - *Given a netlist, a routing grid, a minimum spacing requirement  $d_{min}$  and three colors (RED, GREEN and BLUE), detailed routing with simultaneous color assignment is performed such that the total conflicting wire length and number of stitches are minimized.*

An immediate subproblem is how to perform maze routing for a single net on the routing grid in the presence of a set of previously routed nets. When we route a net, it is desired to compute a path  $p$  with color assignment that produces minimum conflicting wire length and minimum number of stitches. Of course, the wire length of the net, as a conventional metric, also needs to be minimized. Therefore, the weighted sum  $l_w^p + \alpha l_{con}^p + \beta n_s^p$  is a good cost metric to minimize when we route a single net, where  $l_w^p$ ,  $l_{con}^p$  and  $n_s^p$  denote the wire length, the conflicting wire length and the number of stitches produced by the path  $p$  computed, respectively, and  $\alpha$  and  $\beta$  are user defined parameters that specify the relative importance between them. We define the triple patterning aware maze routing problem below.

**Definition 8. Triple Patterning Aware Maze Routing** - *Given a set of previously routed nets together with their color assignment on a routing grid, as well as two pins of a net, the objective is to compute a path  $p$  with color assignment between the two pins such that the weighted sum  $l_w^p + \alpha l_{con}^p + \beta n_s^p$  is minimized.*

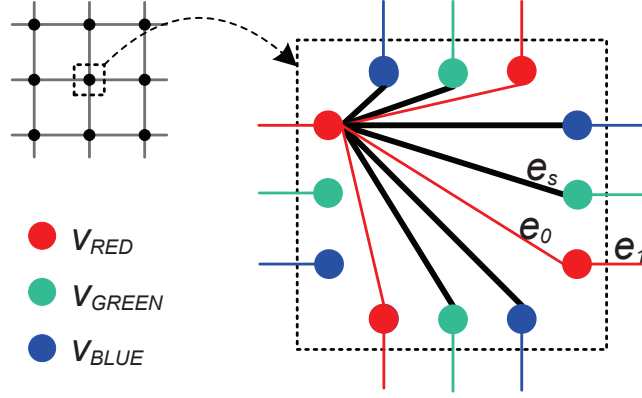


Figure 7.2: Graph model for a non-pin vertex in the routing grid.

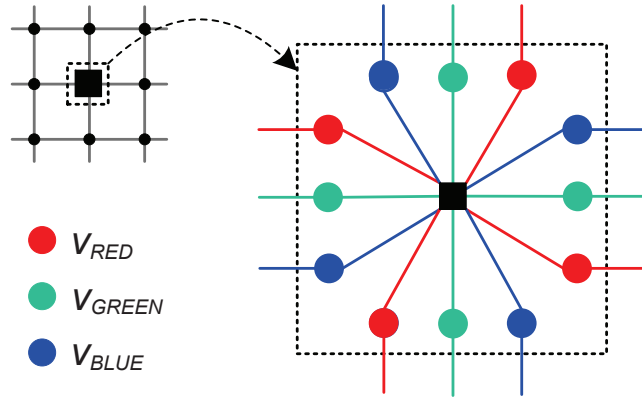


Figure 7.3: Graph model for a pin vertex in the routing grid.

## 7.3 Routing a Single Net

In this section, we propose a graph model that correctly captures the cost of conflicts and stitches, and show that the triple patterning aware maze routing problem can be optimally solved by performing shortest path algorithm on an expanded routing graph constructed using the graph model. We then discuss a practical extension of the proposed graph model to forbid stitch at corner.

### 7.3.1 Triple Patterning Aware Maze Routing

Suppose we are given a routing grid  $G$ , which can be viewed as a routing graph if we regard every intersection of four line segments as a vertex and the short segments between vertices as edges. In the triple patterning aware

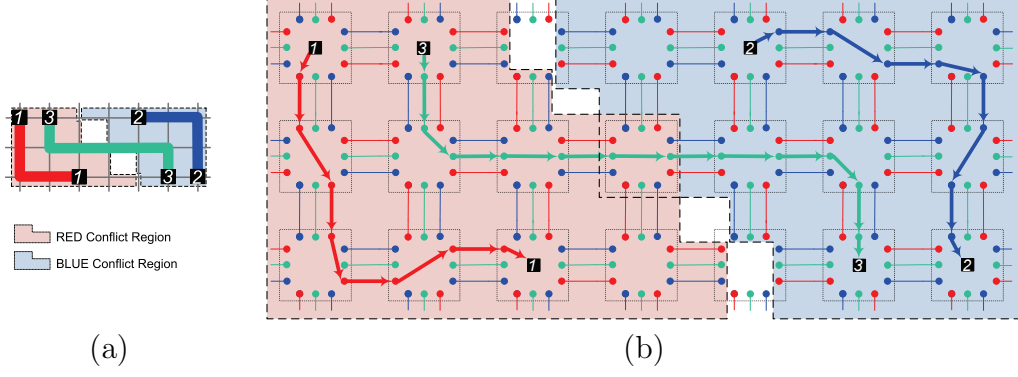


Figure 7.4: (a) Routes on the original routing grid; (b) Routes on the expanded routing graph.

maze routing problem, the cost of wire length and conflicting wire length can be easily captured by assigning cost to the edges of  $G$ . However, the cost of stitches cannot be directly captured. In order to capture the cost of stitches as well, we split each vertex  $v$  of  $G$  into 12 vertices and construct a graph model on them, as shown in Figure 7.2. The detailed construction is described as follows:

- The 12 vertices fall into three categories, namely,  $v_{RED}$ ,  $v_{GREEN}$  and  $v_{BLUE}$ , which correspond to the three colors RED, GREEN and BLUE, respectively. On each of the four boundaries of the graph model, there are one  $v_{RED}$ , one  $v_{GREEN}$  and one  $v_{BLUE}$ , as shown in Figure 7.2. On the two adjacent boundaries of two neighboring graph models, the two vertices of the same color are connected.
- Within the graph model, two vertices are connected by an edge if and only if they are not lying on the same boundary. Note that in Figure 7.2, only the edges adjacent to the vertex  $v_{RED}$  on the left boundary are displayed. This graph model works like a switch box. A route can come into the graph model at any boundary and go out at any other boundary. The color of the route can be changed within the model.
- All the edges can be categorized into three types, namely,  $e_0$ ,  $e_s$  and  $e_1$ . A type  $e_0$  edge with cost 0 connects two vertices of the same color within the graph model. A type  $e_s$  edge with cost  $\beta$  (the cost of a stitch), as indicated by a thick edge in Figure 7.2, connects two vertices of different colors within the graph model. It corresponds to a stitch.

A type  $e_1$  edge connects two same color vertices of two neighboring graph models. It corresponds to an edge in the original routing grid. The base cost of a type  $e_1$  edge is 1, which indicates the cost of the unit wire length. However, when routing on a type  $e_1$  edge causes a coloring conflict, the cost of this edge will be updated to  $(1 + \alpha)$ , where  $\alpha$  is the cost of the unit conflicting wire length. Note that the type  $e_1$  edges are shared by the neighboring graph models.

Note that if a vertex  $v$  in  $G$  is a pin of a net, we will construct the graph model for  $v$  as shown in Figure 7.3 instead, where  $v$  is still split into the 12 vertices, and another vertex representing the pin will be added and connected to the 12 vertices.

We now obtain an expanded routing graph  $G'$  from the original routing grid  $G$  through the construction described above. When we route one net, we simply apply Dijkstra's shortest path algorithm on  $G'$ . Figure 7.4 gives an example for illustration. Figure 7.4(a) displays the routes of three nets on the original routing grid, while Figure 7.4(b) demonstrates the corresponding routes on the expanded routing graph. Suppose, without loss of generality, the net ordering during routing is net 1, net 2, then net 3. Net 1 and net 2 are routed and assigned to RED and BLUE, respectively. The two shaded regions are respectively RED conflict region and BLUE conflict region. The type  $e_1$  edges connecting two  $v_{RED}$  ( $v_{BLUE}$ ) vertices within the RED (BLUE) conflict region will have their cost be updated as  $1 + \alpha$ , indicating that using these edges for routing will cause conflicts. Therefore, when net 3 is routed, the shortest path algorithm will find the path colored GREEN as shown in Figure 7.4.

To show the optimality of our graph expansion based approach, we first show that the shortest path  $p'$  between two pins on  $G'$  must correspond to a path  $p$  between the two pins on  $G$ . Suppose the shortest path  $p'$  on  $G'$  does not correspond to a path  $p$  on  $G$ , and it corresponds to a walk with loop on  $G$  instead. It follows that  $p'$  enters a graph model at least twice on  $G'$ . Let us assume that, w.l.o.g.,  $p'$  enters a graph model twice as shown in Figure 7.5. However, the part of path  $p'$  that produces a loop can always be shortcut by an edge inside the graph model. This indicates that  $p'$  is not a shortest path, which is contradictory to the assumption. Note that the shortest path  $p'$  on  $G'$  also determines the color assignment of its corresponding path  $p$  on

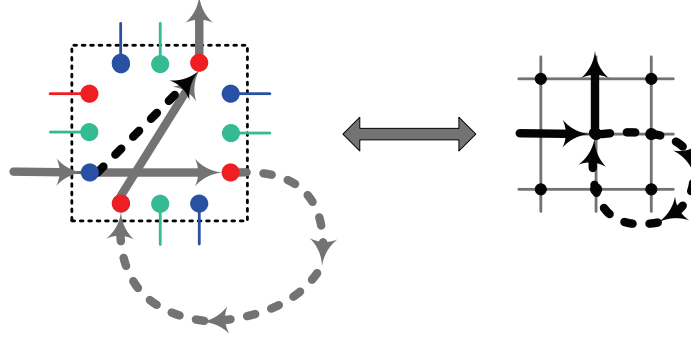


Figure 7.5: A path in the expanded routing  $G'$  may correspond to a walk with loop in the routing grid  $G$ .

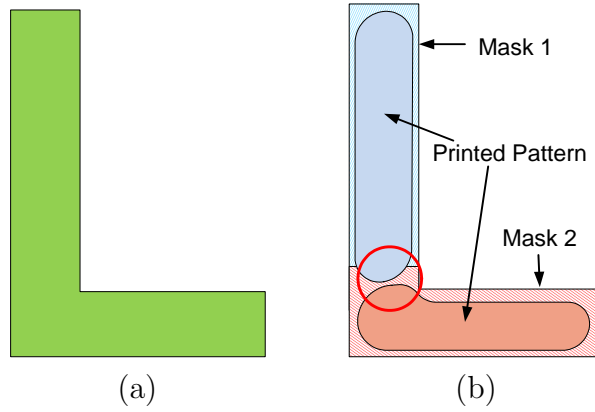


Figure 7.6: (a) Pattern to be printed; (b) Stitching at corner results in significant printability degradation due to overlay error and line-end effect.

$G$ . Then, from our graph model construction and cost setting, it is easy to see that the  $l_w^p + \alpha l_{con}^p + \beta n_s^p$  value of a path  $p$  with color assignment on  $G$  is the same as the cost of its corresponding path  $p'$  on  $G'$ . Thus, the shortest path  $p'$  between two pins on  $G'$  corresponds to a colored path  $p$  of smallest  $l_w^p + \alpha l_{con}^p + \beta n_s^p$  value between the two pins on  $G$ . Based on the above analysis, we can conclude that the triple patterning aware maze routing problem on a routing grid  $G$  can be optimally solved by computing the shortest path between the two pins on the expanded routing graph  $G'$ .

### 7.3.2 Forbidding Stitch at Corner

A routing path can change its color assignment at some point to avoid conflicts, which introduces a stitch. Stitches are sensitive to overlay errors and thus lead to printability degradation. In particular, when a stitch occurs at

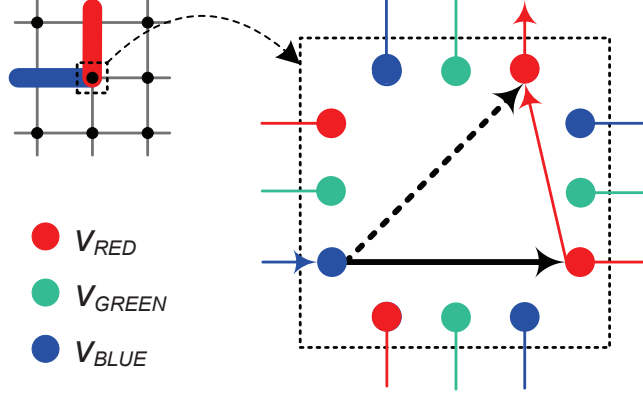


Figure 7.7: Corner stitches cannot be prevented by simply removing the edges that directly generate corner stitches.

a turning point, or corner, of a route, the situation is even worse and the degradation of printability can be much more significant [76], as illustrated in Figure 7.6. Therefore, a stitch at a corner is highly undesirable. In this subsection, we show that our graph model can be extended to disallow corner stitches.

Intuitively, a corner stitch can be prohibited by removing the type  $e_s$  edges that introduce corner stitches in the graph model. For example, in Figure 7.7, the edge connecting  $v_{BLUE}$  on the left boundary and  $v_{RED}$  on the top boundary (shown in dashed line) will be removed. However, this does not work as  $v_{BLUE}$  on the left boundary can still reach  $v_{RED}$  on the top boundary by taking a detour as shown by the solid edges in Figure 7.7. To address this issue, we further split each vertex  $v$  in the graph model into two vertices  $v^{in}$  and  $v^{out}$ .  $v_{RED}$  ( $v_{GREEN}$ ,  $v_{BLUE}$ ) is split into  $v_{RED}^{in}$  ( $v_{GREEN}^{in}$ ,  $v_{BLUE}^{in}$ ) and  $v_{RED}^{out}$  ( $v_{GREEN}^{out}$ ,  $v_{BLUE}^{out}$ ). We also make the edges directed. The edges are coming into the graph model through  $v^{in}$ , and going out of the graph model through  $v^{out}$ . Within the graph model, an edge will be directed from  $v^{in}$  to  $v^{out}$ . The new graph model that disallows corner stitches is shown in Figure 7.8. Note that only the edges adjacent to the vertices on the left boundary are displayed. A minor flaw of using this new graph model is that a loop similar to the one shown in Figure 7.5 may be produced by applying the shortest path algorithm. A path may take a detour outside the graph model to reach  $v_{RED}^{out}$  on the top boundary from  $v_{BLUE}^{in}$  on the left boundary, but this time the detour can no longer be shortcut as there is no connection between the two vertices within the new graph model. However, according to our experiment,

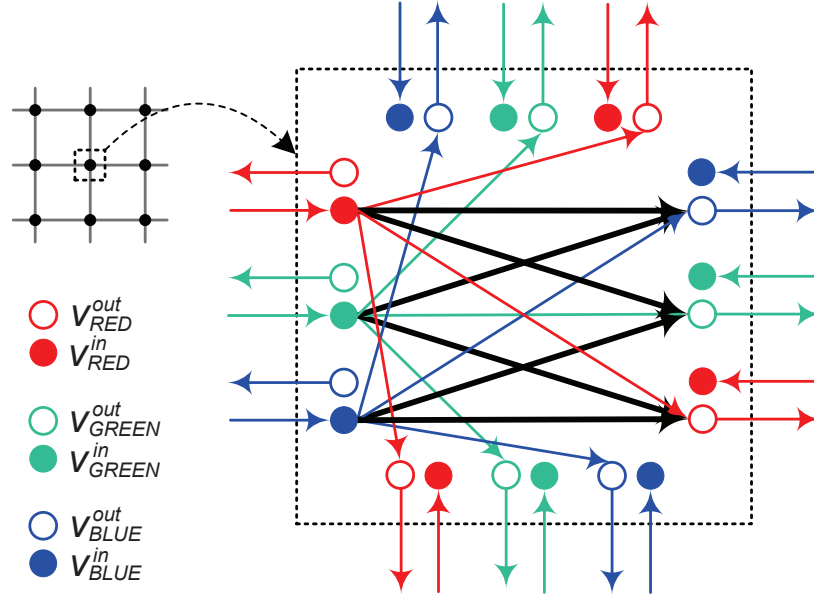


Figure 7.8: Graph model for TPL that disallows stitch at corner.

this situation rarely happens. When it happens, we simply block the place where the detour occurs and run the shortest path algorithm again. This technique turns out to be very effective.

Our proposed graph model is a unified graph model, which can be extended for multiple patterning lithography, and can also be tailored for double patterning lithography. The graph model for DPL is shown in Figure 7.9, which uses two colors *RED* and *BLUE*.

### 7.3.3 Handling Miscellaneous Cost Metrics

In the triple patterning aware maze routing problem, a path  $p$  is evaluated by the weighted sum  $l_w^p + \alpha l_{con}^p + \beta n_s^p$ . The weighted sum is used to update the distance of the vertices during the propagation step in the shortest path algorithm. In some cases, other cost metrics may work better. For example, for some critical nets, conflicting wire length is the dominating factor, the number of stitches is secondary, and its wire length is least important. In this case, a three tuple  $[l_{con}^p, n_s^p, l_w^p]$  is a better cost metric. For two candidate paths  $p_1$  and  $p_2$ ,  $p_1$  is better than  $p_2$  if  $l_{con}^{p_1} < l_{con}^{p_2}$ ; if  $l_{con}^{p_1} = l_{con}^{p_2}$ ,  $n_s^{p_1}$  and  $n_s^{p_2}$  are compared; if  $n_s^{p_1} = n_s^{p_2}$ ,  $l_w^{p_1}$  and  $l_w^{p_2}$  are compared. Our graph model can



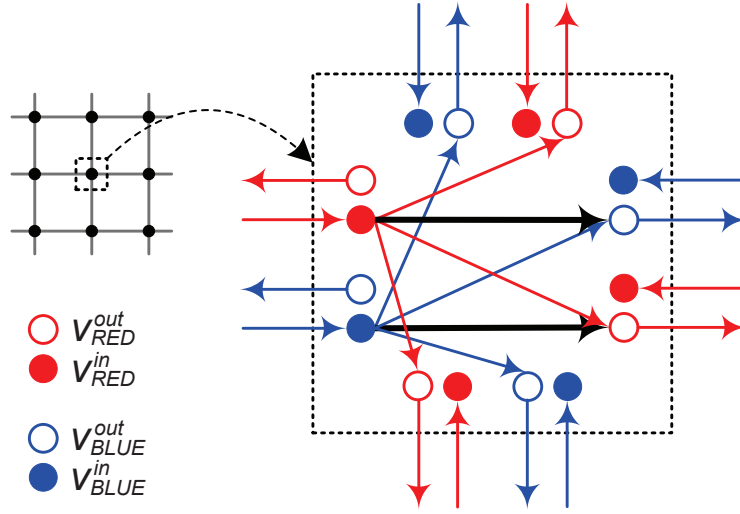


Figure 7.9: Graph model for DPL that disallows stitch at corner.

be easily modified to handle this kind of cost metric. We set the cost of each edge in the graph model to be a three tuple. The cost of a type  $e_0$  edge is set to be  $[0, 0, 0]$ . The cost of a type  $e_s$  edge is set to be  $[0, 1, 0]$ . As for a type  $e_1$  edge, its cost is set to be  $[0, 0, 1]$  if routing on it does not cause a conflict, and its cost is set to be  $[1, 0, 1]$  otherwise. During the propagation step of the shortest path algorithm, a three tuple will be used to update the distance of neighboring vertices. In this way, we are able to find the shortest path  $p$  for a net in terms of  $[l_{con}^p, n_s^p, l_w^p]$ . The via cost can also easily be taken into account. In the expanded routing graph  $G'$ , we simply create a set of via vertices to connect the corresponding graph models of neighboring layers. Let  $n_v^p$  denote the number of vias on path  $p$ , then the three tuple can be changed to four tuple  $[l_{con}^p, n_s^p, n_v^p, l_w^p]$  to take the via cost into account. The via cost can also be added to the weighted sum version.

## 7.4 Overall Routing Scheme

In this section we present our overall routing scheme for the triple patterning aware detailed routing problem. We adopt a negotiated congestion based scheme to resolve the coloring conflicts over iterations of rip-up and reroute/re-color. Since the nets are routed and colored one by one sequentially, the solution obtained within one single pass is not good enough, and

coloring conflicts may exist. The negotiated congestion based scheme is able to dynamically refine the routing and coloring solution over iterations. This scheme can significantly reduce the adverse effect of improper net ordering. We also propose a way to balance the features over the masks, which is beneficial for manufacturing. Our overall routing scheme also works for double patterning aware detailed routing.

#### 7.4.1 Negotiated Congestion based Scheme to resolve Coloring Conflicts

The negotiated congestion based routing scheme [7] has been widely used in FPGA routing and global routing to resolve routing congestions. In this routing scheme, routability is achieved by forcing all the nets to negotiate for a resource and thereby determine which net needs the resource most. Some nets may use shared resources that are in high demand if all alternative routes utilize resources in even higher demand; other nets will tend to spread out and use resources in lower demand. All the nets are iteratively rerouted until no more resources are shared.

We adapt this negotiated congestion based scheme to resolve coloring conflicts in our triple patterning aware detailed routing problem. We let the nets negotiate for the color assignment by adding a history cost to the type  $e_1$  edges in the expanded routing graph  $G'$ . The cost of a type  $e_1$  edge is computed by the following formula:

$$cost(e_1) = 1 + \alpha \times (isConflict? h_c : 0),$$

where *isConflict* is a Boolean variable indicating if this edge lies in the conflict region produced by the previously routed nets, and  $h_c$  denotes the history cost. This means that if routing on this type  $e_1$  edge causes a conflict, the cost of this edge will be set to  $1 + \alpha \times h_c$ ; otherwise, the cost will be set to 1.  $h_c$  is initialized as 1.

This scheme works as follows. We start with a global routing solution without color assignment. In the initial iteration, each net is rerouted on the expanded routing graph  $G'$ . When the initial iteration terminates, the routes on  $G'$  provide a color assignment of all the nets. Note that during routing one net, the access to the graph models occupied by other nets is

denied, so that no crossing will be generated and routability is guaranteed. If coloring conflicts exist, iterations of rip-up and reroute will be performed. When a net  $i$  is rerouted, we first remove its current route, as well as the conflict region(s) it produces. If a type  $e_1$  edge was lying in net  $i$ 's conflict region of the same color, its flag *isConflict* will be updated as *false* since the conflict region is now gone. The shortest path algorithm is then performed to compute a new path for net  $i$ , and the conflict region(s) it produces will be updated. If a type  $e_1$  edge falls into a conflict region representing the same color, its flag *isConflict* will be set as *true*. In addition, if the path of this net causes coloring conflicts with previously routed nets, the type  $e_1$  edges that are responsible for the conflicts will have their history cost  $h_c$  incremented by 1. In this way, the regions with coloring conflicts grow more expensive over iterations, and those nets with more options will tend to choose alternative routes or colors in subsequent iterations, so that the conflicts can potentially be resolved. In our implementation, this procedure will terminate when either no more conflicts exist or enough iterations of rerouting have been performed. Our experimental results show that this scheme is very effective in resolving conflicts.

The example in Figure 7.10 demonstrates how this negotiated congestion based scheme works. There are 6 nets, where net 1-5 are previously routed with color assignment, and net 6 is the current net to be routed, as shown in Figure 7.10(a). It is easy to see that net 6 will cause conflict with other nets no matter what color it is assigned. Net 6 is colored in RED by the shortest path algorithm, which causes conflicts with net 4, as shown in Figure 7.10(b). As a result, the type  $e_1$  edges that are responsible for the conflicts have their history cost  $h_c$  incremented. Therefore, in the next iteration, net 4 is rerouted and colored in BLUE, so that the conflicts are effectively resolved, as shown in Figure 7.10(c).

## 7.4.2 Balancing Features on Three Masks

TPL provides three masks to accommodate the features. Balancing the features on the three masks ensures that each mask is fully utilized, so that none of the masks is unnecessarily dense. This helps the printability enhancement during manufacturing. We develop a heuristic that can effectively control

the balancing on the fly. During routing, we maintain three variables  $l_{RED}$ ,  $l_{GREEN}$  and  $l_{BLUE}$ , which respectively keep track of the wire length colored in RED, GREEN and BLUE. The cost of the type  $e_1$  edges will be adjusted according to the current distribution of the total wire length on the three masks. For example, when relatively more wires are assigned to GREEN, the cost of the type  $e_1$  edges of *GREEN* color should be increased, so that the router tends to favor the routes using other colors. In our implementation, we let  $l_{RED}$  be the reference. The cost of the type  $e_1$  edges of GREEN color is scaled by  $\frac{l_{GREEN}}{l_{RED}}$ , and the cost of those of BLUE color is scaled by  $\frac{l_{BLUE}}{l_{RED}}$ . This technique is shown to be effective through the experiments.

The pseudocode of our overall routing scheme is listed below.

**ALGORITHM TPL AWARE DETAILED ROUTING( $G, N$ ):**  
Construct the expanded routing graph  $G'$  from  $G$ ;  
Set  $h_c$  as 1 for all the type  $e_1$  edges;  
**for** each net  $i$  in  $N$  **do**      //initial iteration  
    Reroute net  $i$  using shortest path algorithm;  
    Update flag *isConflict* for the affected  $e_1$  edges;  
    Update  $l_{RED}$ ,  $l_{GREEN}$  and  $l_{BLUE}$ ;  
**while**  $\exists$  conflicts **do**  
    **for** each net  $i$  in  $N$  **do**  
        Remove the route of net  $i$ ;  
        Update flag *isConflict* for the affected  $e_1$  edges;  
        Update  $l_{RED}$ ,  $l_{GREEN}$  and  $l_{BLUE}$ ;  
        Reroute net  $i$  using shortest path algorithm;  
        Update flag *isConflict* for the affected  $e_1$  edges;  
        Increment  $h_c$  for the conflicting  $e_1$  edges;  
        Update  $l_{RED}$ ,  $l_{GREEN}$  and  $l_{BLUE}$ ;

## 7.5 Experimental Results

We implement a TPL aware detailed router as well as a DPL version, and compare their performance in 14nm node on two sets of benchmarks. The graph model that disallows corner stitches is adopted for both triple patterning and double patterning. Our program is implemented in C++ and all the

Table 7.1: Comparison of TPL and DPL in 14nm technology

Test Cases	#Net	Size ( $\mu m^2$ )	Init Con. WL( $\mu m$ )		#Iter		Con. WL		#Stitch		#Via		Wirelength ( $\mu m$ )		Runtime (s)	
			DP	TP	DP	TP	DP	TP	DP	TP	DP	TP	DP (R:B)	TP (R:G:B)	DP	TP
test1	1K	31.36	7.39	0.28	16	3	0	0	49	0	648	476	354(1:1.03)	354(1:1.005:0.99)	57	17
test2	2K	70.56	9.07	0.39	9	3	0	0	96	0	1076	750	723(1:1.04)	723(1:1.016:1.0007)	143	68
test3	4K	165.9	8.23	0.28	4	3	0	0	185	0	1622	878	1444(1:1.02)	1461(1:1.02:1.04)	273	301
test4	6K	245.9	12.15	0.56	16	2	0	0	247	0	2252	1496	2130(1:1.012)	2162(1:0.97:1.003)	2431	438
test5	8K	321.1	19.08	0.67	7	3	0	0	365	0	3426	2036	2873(1:0.999)	2911(1:1.01:1.02)	1790	1194
test6	10K	384.2	30.41	2.13	15	4	0	0	419	1	4452	2782	3593(1:1.016)	3625(1:0.998:1.01)	5673	2210
Avg			14.56	0.716	11.2	3	0	0	227	0.17	2246	1403	1852	1873	1728	705
Diff			1	-95%	1	-73%	0	0	1	-99.9%	1	-37%	1	+1.13%	1	-59%

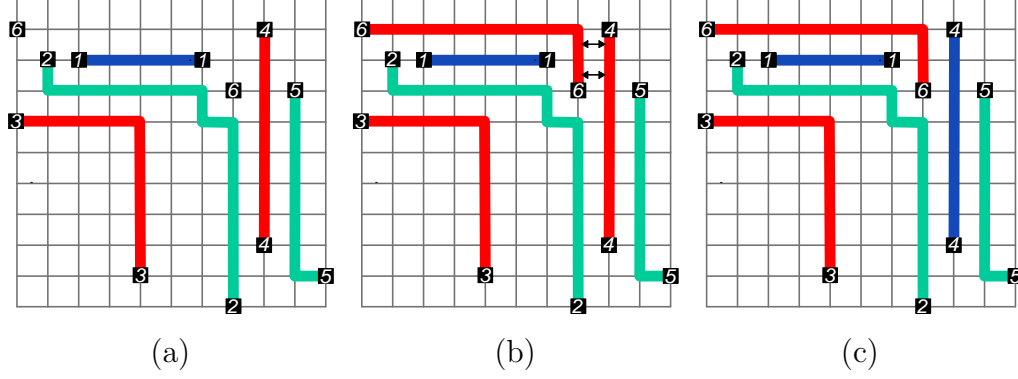


Figure 7.10: An example using the negotiated congestion based scheme to handle conflicts. (a) Nets 1-5 are routed and colored, and net 6 is not routed yet; (b) Net 6 is routed in red, causing conflicts with net 4; (c) In the next iteration, net 4 is rerouted in blue due to the penalty exerted on the red edges.

experiments are performed on a Linux machine with 2.0GHz CPU and 2GB RAM.

We first compare the two routers on a set of benchmarks derived from industrial data. The result of the comparison is shown in Table 7.1. The column “Init Con. WL” shows the conflicting wire length after the initial iteration. The column “#iter” shows the total number of rerouting iterations performed. The column “Con. WL” reports the final conflicting wire length. We can see that our negotiated congestion based routing scheme is very effective in resolving conflicts. All the conflicts can be resolved within a few iterations for both DPL and TPL. The column “#Stitch” and the column “#Via” report the number of stitches and the number of vias in the final solution, respectively. We can see that TPL, with an extra mask to accommodate the features, can remove almost all of the stitches (99.9% on average) that are necessary for DPL. TPL can also reduce the number of vias by 37% on average compared with DPL. The column “Wirelength” reports the total wire length together with the ratio of the wire length on each mask. The results show that our approach can balance the utilization of the masks very well. We then compare the two routers in terms of ability to resolve conflicts on a set of denser benchmarks. The result is displayed in Table 7.2. The maximum number of rerouting iterations is set to be 30. We can see from the result that the double patterning router generates a lot more conflicts from the beginning and cannot resolve all of them after 30 iterations, while

Table 7.2: Comparison of TPL and DPL in resolving conflicts

Test Cases	#Net	Size ( $\mu m^2$ )	Init Con. WL( $\mu m$ )		#Iter		Con. WL( $\mu m$ )	
			DP	TP	DP	TP	DP	TP
test1d	1.2K	31.36	18.47	0.82	30	3	7.68	0
test2d	2.5K	70.56	24.50	1.15	30	5	8.35	0
test3d	5K	165.9	28.40	0.96	30	4	6.24	0
test4d	7.5K	245.9	48.92	1.32	30	4	12.46	0
test5d	10K	321.1	71.26	1.89	30	5	20.79	0
test6d	12K	384.2	87.41	4.26	30	8	37.61	0
Avg			46.49	1.73	30	4.83	15.52	0
Diff			1	-96%	1	-84%	1	-100%

the triple patterning router can easily bring the conflicting wire length down to zero within a small number of iterations.

By comparing DPL and TPL under the same printing condition, we would like to answer the multiple choice question about which patterning technique to use in 14nm technology node. During the experiment and with help of our proposed algorithm, we have set up the comparison between DPL and TPL fairly enough: (1) the same test benches; (2) the same technology node setup; (3) the routing-coloring co-optimized designs for both DPL and TPL. Note that since the stitches can be almost completely avoided (in Table 7.1), the overlay in TPL will not be as harmful as in DPL where stitches are commonly seen. From the comparison (Table 7.1 and Table 7.2), we can see that choosing between DPL and TPL in 14nm node is intrinsically a trade-off between mask/process cost and printability. In the circumstances where stitches are highly likely to cause yield loss and the extra mask cost from TPL is still affordable, with the help of our proposed algorithm, the solution of TPL with simultaneous routing and coloring would be a wise choice in 14nm technology node.

## 7.6 Concluding Remarks

In this chapter, the problem of TPL aware routing is studied. A unified graph model is proposed to solve the maze routing problem in the context of multiple patterning lithography. An overall routing scheme is developed to resolve the coloring conflicts and balance the utilization of the masks. Triple patterning aware routing is compared with double patterning aware routing in 14nm node. The results show that TPL can resolve the conflicts more easily and significantly reduce the number of stitches compared with DPL.

# CHAPTER 8

## CONCLUSIONS

In this dissertation, we have studied modern routing problems. Topics that have been covered in our study include: PCB routing, graphene routing and its reliability, as well as TPL aware routing.

We first focused on bus-based PCB routing. In Chapter 2, we introduced the Rectangle Escape Problem (REP), which originates in PCB bus escape routing. We proved that REP is NP-complete, and proposed a 4-approximation algorithm by using linear programming relaxation and rounding technique. This algorithm was also shown to work for weighed REP and simultaneous REP. Our algorithm was implemented and tested on a set of industrial PCB bus escape routing cases, and the results showed that an optimal solution can be obtained within several seconds for each test case. In Chapter 4 we further studied the layer assignment problem of bus escape routing. We proposed a novel branch-and-bound based optimal layer assignment algorithm. The practical consecutive assignment constraints were addressed and handled. This is the first optimal algorithm proposed in literature for this problem. Experimental results on industrial data validated our approach.

We then studied net-based PCB routing. In Chapter 3, we proposed an underlying routing graph which correctly models the routing resources of the pin grids on board, and then built a Negotiated Congestion based Escape Router (NCER) by applying the negotiated congestion routing scheme on the constructed routing graph. Through experiments, we observed that our NCER performs comparably to the Cadence Allegro PCB router. Moreover, the two routers exhibited complementary behaviors. Combining them together can greatly improve the routability. This observation certainly opens more opportunities in the research of PCB routing. Successful routing techniques from other areas (like FPGA routing and IC routing) can be adapted to PCB routing despite its unique features (planar routing, strict length re-



quirements, grid structure, etc.).

We also studied GNR routing together with its routing reliability. In Chapter 5, we formulated GNR routing as a minimum hybrid-cost shortest path problem. The underlying routing grid is a triangular mesh due to the properties of metallic GNRs. In order to model this hybrid-cost problem, we applied graph expansion on the triangular mesh and then introduced a shortest red-black path problem. We also proposed an algorithm that optimally solved the shortest red-black path problem and therefore optimally solved the minimum hybrid-cost shortest path problem. We fitted our algorithm into a negotiated congestion based routing scheme and tested it by experiment. The experimental results showed that our routing algorithm effectively solved the hybrid-cost routing problem. In Chapter 6, the routing reliability problem was introduced and studied. We analyzed this problem and found that it is theoretically hard. We tackled this problem by dividing it into stages: (1) generation of candidate redundancy segment via min-cost max-flow; (2) optimal selection among the candidates by dynamic programming. The proposed approach was tested on both general graphs and grid graphs, and promising results were obtained. The survival probability of the s-t connection can be increased from 67.93% to 94.17% on average, with a reasonable overhead of routing resources. The results also showed that our proposed approach is very efficient, even large test cases can be accomplished within one second. This makes it possible to integrate our algorithm flow into industrial routers to address the reliability issue.

We finally studied the problem of TPL aware routing in Chapter 7. We first proposed a graph model that correctly models the cost of conflicts and stitches in TPL. By replacing each vertex in the routing grid with the graph model and performing shortest path algorithm on the expanded graph, the optimal path with mask/color assignment for one net can be computed, in presence of previously routed and colored nets. Our proposed graph model is a unified model that can be extended to handle multiple patterning and can also be tailored for double patterning. We then developed a negotiated congestion based scheme to resolve conflicts. Experimental results showed that this scheme is very effective and all conflicts can be resolved within a small number of iterations in our test cases. We also implemented a double patterning aware detailed router by adapting our graph model for double patterning, and compared it with the triple patterning version in 14nm node.

Experimental results showed that, using TPL, the conflicts can be resolved much more easily and the stitches can be significantly reduced in contrast to DPL.

## REFERENCES

- [1] Fujitsu, “IC package, fujitsu microelectronics limited.” [Online]. Available: [www.fujitsu.com/downloads/MICRO/fma/pdf/a810000114e.pdf](http://www.fujitsu.com/downloads/MICRO/fma/pdf/a810000114e.pdf)
- [2] D. Brooks, *Signal Integrity Issues and Printed Circuit Board Design*. Prentice Hall, 2003.
- [3] J. C. Whitaker, *The Electronics Handbook (2nd Edition)*. CRC Press, 2005.
- [4] M. Cho, K. Lu, K. Yuan, and D. Z. Pan, “Boxrouter 2.0: architecture and implementation of a hybrid and robust global router,” in *ICCAD '07: Proceedings of the 2007 IEEE/ACM International Conference on Computer-Aided Design*, 2007, pp. 503–508.
- [5] M. M. Ozdal and M. D. F. Wong, “Archer: a history-driven global routing algorithm,” in *ICCAD '07: Proceedings of the 2007 IEEE/ACM International Conference on Computer-Aided Design*, 2007, pp. 488–495.
- [6] J. A. Roy and I. L. Markov, “High-performance routing at the nanometer scale,” in *ICCAD '07: Proceedings of the 2007 IEEE/ACM International Conference on Computer-Aided Design*, 2007, pp. 496–502.
- [7] L. McMurchie and C. Ebeling, “Pathfinder: a negotiation-based performance-driven router for FPGAs,” in *FPGA '95: Proceedings of the 1995 ACM third International Symposium on Field-Programmable Gate Arrays*, 1995, pp. 111–117.
- [8] A. K. Geim, “Graphene: status and prospects,” *Science*, no. 5934, pp. 1530–1534, 2009.
- [9] K. A. Ritter and J. W. Lyding, “The influence of edge structure on the electronic properties of graphene quantum dots and nanoribbons,” *Nature Materials*, no. 3, pp. 235–242, Mar. 2009.
- [10] G. E. Bailey, A. Tritchkov, J.-W. Park, L. Hong, V. Wiaux, E. Hendrickx, S. Verhaegen, P. Xie, and J. Versluijs, “Double pattern EDA solutions for 32nm hp and beyond,” in *SPIE*, vol. 6521, 2007.

- [11] J. Huckabay, W. Staud, R. Naber, A. Oosten, P. Nikolski, S. Hsu, R. J. Socha, M. V. Dusa, and D. Flagello, "Process results using automatic pitch decomposition and double patterning technology (DPT) at  $k_{1eff} < 0.20$ ," in *SPIE*, vol. 6349, 2006.
- [12] Y. Inazuki, N. Toyama, T. Nagai, T. Sutou, Y. Morikawa, H. Mohri, N. Hayashi, M. Drapeau, K. Lucas, and C. Cork, "Decomposition difficulty analysis for double patterning and the impact on photomask manufacturability," in *SPIE*, vol. 6925, 2008.
- [13] V. Wiaux, S. Verhaegen, S. Cheng, F. Iwamoto, P. Jaenen, M. Maenhoudt, T. Matsuda, S. Postnikov, and G. Vandenberghe, "Split and design guidelines for double patterning," in *SPIE*, vol. 6924, 2008.
- [14] M. Ozdal, M. Wong, and P. Honsinger, "An escape routing framework for dense boards with high-speed design constraints," in *Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on*, nov. 2005, pp. 759 – 766.
- [15] H. Harrer, H. Pross, T.-M. Winkel, W. D. Becker, H. I. Stoller, M. Yamamoto, S. Abe, B. J. Chamberlin, and G. A. Katopis, "First- and second-level packaging for the IBM eserver z900," *IBM J. Res. Dev.*, vol. 46, no. 4-5, pp. 397–420, July 2002.
- [16] T.-M. Winkel, W. D. Becker, H. Harrer, H. Pross, D. Kaller, B. Garben, B. J. Chamberlin, and S. A. Kuppinger, "First- and second-level packaging of the z990 processor cage," *IBM J. Res. Dev.*, vol. 48, no. 3-4, pp. 379–394, May 2004.
- [17] H. Kong, Q. Ma, T. Yan, and M. D. Wong, "An optimal algorithm for finding disjoint rectangles and its application to PCB routing," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, June 2010, pp. 212 –217.
- [18] H. Kong, T. Yan, and M. Wong, "Automatic bus planner for dense PCBs," in *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, July 2009, pp. 326 –331.
- [19] H. Kong, T. Yan, M. Wong, and M. Ozdal, "Optimal bus sequencing for escape routing in dense pcbs," in *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, Nov. 2007, pp. 390 –395.
- [20] H. Kong, T. Yan, and M. Wong, "Optimal simultaneous pin assignment and escape routing for dense PCBs," in *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, Jan. 2010, pp. 275 –280.

- [21] T. Yan and M. Wong, “BSG-route: A length-constrained routing scheme for general planar topology,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 11, pp. 1679–1690, Nov. 2009.
- [22] Q. Ma, H. Kong, M. Wong, and E. Young, “A provably good approximation algorithm for rectangle escape problem with application to PCB routing,” in *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, Jan. 2011, pp. 843–848.
- [23] Q. Ma and M. D. F. Wong, “NP-completeness and an approximation algorithm for rectangle escape problem with application to PCB routing,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, Sept 2012.
- [24] D. T. Lee, “Maximum clique problem of rectangle graphs,” *Advances in Computing Research*, vol. 1, pp. 91–107, 1983.
- [25] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.
- [26] 2010. [Online]. Available: <http://www.gurobi.com>
- [27] W.-T. Chan and F. Y. L. Chin, “Efficient algorithms for finding the maximum number of disjoint paths in grids,” *J. Algorithms*, vol. 34, no. 2, pp. 337–369, 2000.
- [28] J.-W. Fang, I.-J. Lin, P.-H. Yuh, Y.-W. Chang, and J.-H. Wang, “A routing algorithm for flip-chip design,” in *ICCAD ’05: Proceedings of the 2005 IEEE/ACM International Conference on Computer-Aided Design*, 2005, pp. 753–758.
- [29] R. Wang, R. Shi, and C.-K. Cheng, “Layer minimization of escape routing in area array packaging,” in *ICCAD ’06: Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design*, 2006, pp. 815–819.
- [30] M.-F. Yu and W. W.-M. Dai, “Single-layer fanout routing and routability analysis for ball grid arrays,” in *ICCAD ’95: Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design*, 1995, pp. 581–586.
- [31] T. Yan and M. D. F. Wong, “A correct network flow model for escape routing,” in *DAC ’09: Proceedings of the 46rd ACM/IEEE Design Automation Conference*, 2009, pp. 332 – 335.

- [32] L. Luo and M. D. F. Wong, "Ordered escape routing based on Boolean satisfiability," in *ASP-DAC '08: Proceedings of the 2008 Asia and South Pacific Design Automation Conference*, 2008, pp. 244–249.
- [33] Y. Tomioka and A. Takahashi, "Monotonic parallel and orthogonal routing for single-layer ball grid array packages," in *ASP-DAC '06: Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, 2006, pp. 642–647.
- [34] J.-W. Fang, C.-H. Hsu, and Y.-W. Chang, "An integer linear programming based routing algorithm for flip-chip design," in *DAC '07: Proceedings of the 44th Annual Design Automation Conference*, 2007, pp. 606–611.
- [35] M. M. Ozdal, M. D. F. Wong, and P. S. Honsinger, "Simultaneous escape-routing algorithms for via minimization of high-speed boards," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 84–95, 2008.
- [36] M. M. Ozdal and M. D. F. Wong, "Simultaneous escape routing and layer assignment for dense pcbs," in *ICCAD '04: Proceedings of the 2004 IEEE/ACM International Conference on Computer-Aided Design*, 2004, pp. 822–829.
- [37] M. M. Ozdal and M. D. F. Wong, "Length-matching routing for high-speed printed circuit boards," in *ICCAD '03: Proceedings of the 2003 IEEE/ACM International Conference on Computer-Aided Design*, 2003, p. 394.
- [38] Q. Ma, T. Yan, and M. Wong, "A negotiated congestion based router for simultaneous escape routing," in *Quality Electronic Design (ISQED), 2010 11th International Symposium on*, march 2010, pp. 606 –610.
- [39] T. Yan and M. D. F. Wong, "Theories and algorithms on single-detour routing for untangling twisted bus," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 3, pp. 1–21, 2009.
- [40] T. Yan, H. Kong, and M. Wong, "Optimal layer assignment for escape routing of buses," in *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, Nov. 2009, pp. 245 –248.
- [41] Q. Ma, E. Young, and M. Wong, "An optimal algorithm for layer assignment of bus escape routing on PCBs," in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, June 2011, pp. 176 –181.

- [42] X. Tang and D. Wong, “Fast-SP: a fast algorithm for block placement based on sequence pair,” in *Design Automation Conference, 2001. Proceedings of the ASP-DAC 2001. Asia and South Pacific*, 2001, pp. 521–526.
- [43] M. R. Garey, D. S. Johnson, C. L. Miller, and C. H. Papadimitriou, “The complexity of coloring circular arcs and chords,” *SIAM Journal of Algebraic Discrete Methods*, vol. 1, no. 2, pp. 216–227, 1980.
- [44] V. Stix, “Target-oriented branch and bound method for global optimization,” in *Journal of Global Optimization*, pp. 261–277, 2003.
- [45] M. Stoer and F. Wagner, “A simple min-cut algorithm,” *Journal of the ACM*, vol. 44, no. 4, pp. 585–591, 1997.
- [46] Y.-M. Lin, K. A. Jenkins, A. Valdes-Garcia, J. P. Small, D. B. Farmer, and P. Avouris, “Operation of graphene transistors at gigahertz frequencies,” *Nano Letters*, no. 1, pp. 422–426, 2009.
- [47] T. Yan, Q. Ma, S. Chilstedt, M. Wong, and D. Chen, “Routing with graphene nanoribbons,” in *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, Jan. 2011, pp. 323–329.
- [48] J.-H. Chen, C. Jang, S. Xiao, M. Ishigami, and M. S. Fuhrer, “Intrinsic and extrinsic performance limits of graphene devices on SiO<sub>2</sub>,” *Nature Nanotechnology*, vol. 3, no. 4, pp. 206–209, Apr. 2008.
- [49] H. Li, C. Xu, N. Srivastava, and K. Banerjee, “Carbon nanomaterials for next-generation interconnects and passives: Physics, status, and prospects,” in *IEEE Transactions on Electron Devices*, no. 9, pp. 1799–1821, Sept 2009.
- [50] T. Dürkop, S. A. Getty, E. Cobas, and M. S. Fuhrer, “Extraordinary mobility in semiconducting carbon nanotubes,” *Nano Letters*, vol. 4, no. 1, pp. 35–39, Jan. 2004.
- [51] K. Nakada, M. Fujita, G. Dresselhaus, and M. S. Dresselhaus, “Edge state in graphene ribbons: Nanometer size effect and edge shape dependence,” *Physical Review B*, vol. 54, no. 24, pp. 17 954–17 961, Dec. 1996.
- [52] Y.-W. Son, M. L. Cohen, and S. G. Louie, “Energy gaps in graphene nanoribbons,” *Phys. Rev. Lett.*, vol. 97, no. 21, p. 216803, 2006.
- [53] S. G. Mark P. Levendorf, Carlos S. Ruiz-Vargas and J. Park, “Transfer-free batch fabrication of single layer graphene transistors,” *Nano Letters*, no. 12, pp. 4479–4483, 2009.

- [54] Q. Yan, B. Huang, J. Yu, F. Zheng, J. Zang, J. Wu, B.-L. Gu, F. Liu, and W. Duan, "Intrinsic current-voltage characteristics of graphene nanoribbon transistors and effect of edge doping," *Nano Letters*, vol. 7, no. 6, pp. 1469–1473, 2007.
- [55] Z. Z. Zhang, Z. H. Wu, K. Chang, and F. M. Peeters, "Resonant tunneling through s- and u-shaped graphene nanoribbons," *Nanotechnology*, vol. 20, no. 41, p. 415203, 2009.
- [56] B. Huang, Q.-M. Yan, Z.-Y. Li, and W.-H. Duan, "Towards graphene nanoribbon-based electronics," *Frontiers of Physics in China*, vol. 4, no. 3, pp. 269–279, Sep. 2009.
- [57] T. Ragheb and Y. Massoud, "On the modeling of resistance in graphene nanoribbon (GNR) for future interconnect applications," 2008, pp. 593–597.
- [58] D. A. Areshkin and C. T. White, "Building blocks for integrated graphene circuits," *Nano Letters*, no. 11, pp. 3253–3259, 2007.
- [59] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *Micro, IEEE*, vol. 25, no. 6, pp. 10 – 16, nov.-dec. 2005.
- [60] A. Kahng, B. Liu, and I. Mandoiu, "Nontree routing for reliability and yield improvement [IC layout]," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 23, no. 1, pp. 148 – 156, Jan. 2004.
- [61] Q. Ma, Z. Xiao, and M. D. Wong, "Algorithmic study on the routing reliability problem," in *Quality Electronic Design (ISQED), 2012 13th International Symposium on*, March 2012, pp. 483 –488.
- [62] J. S. Provan and M. O. Ball, "The complexity of counting cuts and of computing the probability that a network remains connected," *SIAM Journal on Computing*, vol. 12, no. 4, pp. 777–788, 1983.
- [63] L. Valiant, "The complexity of enumeration and reliability problems," *SIAM Journal on Computing*, vol. 8, pp. 410–421, 1979.
- [64] [Online]. Available: [http://en.wikipedia.org/wiki/Series-parallel\\_graph](http://en.wikipedia.org/wiki/Series-parallel_graph)
- [65] H. M. Aboelfotoh and C. J. Colbourn, "Series-parallel bounds for the two-terminal reliability problem," *ORSA Journal on Computing*, vol. 1, no. 4, pp. 209–222, 1989.
- [66] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Application*. Prentice Hall, 1993.



- [67] A. Kahng, C.-H. Park, X. Xu, and H. Yao, "Layout decomposition for double patterning lithography," in *Computer-Aided Design, 2008. ICCAD 2008. IEEE/ACM International Conference on*, Nov. 2008, pp. 465–472.
- [68] Y. Xu and C. Chu, "GREMA: Graph reduction based efficient mask assignment for double patterning technology," in *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, Nov. 2009, pp. 601–606.
- [69] X. Tang and M. Cho, "Optimal layout decomposition for double patterning technology," in *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, nov. 2011, pp. 9–13.
- [70] [Online]. Available: <http://www.itrs.net>
- [71] B. Yu, K. Yuan, B. Zhang, D. Ding, and D. Pan, "Layout decomposition for triple patterning lithography," in *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, Nov. 2011, pp. 1–8.
- [72] M. Cho, Y. Ban, and D. Pan, "Double patterning technology friendly detailed routing," in *Computer-Aided Design, 2008. ICCAD 2008. IEEE/ACM International Conference on*, nov. 2008, pp. 506–511.
- [73] Y.-H. Lin and Y.-L. Li, "Double patterning lithography aware gridless detailed routing with innovative conflict graph," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, June 2010, pp. 398–403.
- [74] J. Sun, Y. Lu, H. Zhou, and X. Zeng, "Post-routing layer assignment for double patterning," in *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, Jan. 2011, pp. 793–798.
- [75] Q. Ma, H. Zhang, and M. D. Wong, "Triple patterning aware routing and its comparison with double patterning aware routing in 14nm technology," in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, June 2012.
- [76] D. Abercrombie, P. Lacour, O. El-Sewefy, A. Volkov, E. Levine, K. Arb, C. Reid, Q. Li, and P. Ghosh, "Double patterning from design enablement to verification," in *SPIE*, vol. 8166.