2023-05-01

An Efficient Cooperation between Routing and Placement with Technology Node Constraints

Aghaeekiasaraee, Erfan

UNIVERSITY OF CALGARY

An Efficient Cooperation between Routing and Placement with Technology Node Constraints

by

Erfan Aghaeekiasaraee

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA
MAY, 2023

Abstract

Traditionally, the placement and routing stages of the physical design are performed separately. However, because of the additional complexities arising in advanced technology nodes, they have become more inter-dependent. As a result, finding ways to improve the cooperation between routing and placement has become an important topic in Electronic Design Automation (EDA).

In this thesis, a framework that incorporates cooperation between routing and placement with the existence of technology node constraints is proposed and developed. The core of this framework is Cooperation Routing and Placement (CRP) engine, which includes techniques to combine routing and placement. The proposed engine is tested on the ACM/IEEE International Symposium on Physical Design (ISPD) 2018 and 2019 contest benchmarks. In the proposed engine, to generate candidate placements, an Integer Linear Programming (ILP)-based Detailed Placement (ILP-DP) engine is developed, that can generate multiple high-quality legalized positions for each cell. Also, due to the complexity of routing topology with technology node constraints, a net classification technique is used to route nets according to net characteristics. Two routing algorithms including AStar and PatternRoute are used to route and estimate the cost of each cell's movement. In addition to that, since runtime is one of the main challenges in the cooperation between routing and placement, two caching techniques called Cost and Net caching are proposed.

Thanks to the Cost caching technique, the global routing runtime compared with state-of-the-art improved by 28.56% on average. Also, by using the Net caching technique a parallel approach is developed that speedup 2.6 times compared to a sequential approach. Numerical results show that the proposed framework by moving 0.7% of cells can improve the detailed routing score by 0.3% on average in the presence of technology node constraints. The proposed engine also shows how only improving single objective wirelength estimation techniques like Half-Perimeter WireLength (HPWL) in the placement step can degrade the quality of the detailed routing solution. This shows the importance of using 3D routing feedback during the placement. The proposed engine can be employed as an add-on to the physical design flow between the global routing and detailed routing steps to improve the quality of the detailed routing solution.

Preface

This thesis is the result of my hard work, spanning over four years, devoted to understanding the intricacies of problem-solving in EDA. I hope that not only will you find it an enjoyable read, but that it will also serve as a valuable source of knowledge that I have gained during this time.

Acknowledgements

I would like to begin by expressing my sincerest gratitude to my supervisor, Dr. Laleh Behjat, for her unwavering support throughout my Ph.D studies. Her support and guidance have been invaluable in the research and writing of this thesis. I also extend my thanks to my co-supervisor, Dr. David Westwick, for his insightful comments and guidance. In addition, I would like to express my gratitude to Dr. José Luís Güntzel and his team, including Tiago Augusto Fontana, Renan Netto, and Sheiny Fabre Almeida, at the University of Federal University of Santa Catarina (UFSC) for their help during my Ph.D studies and research, and for sharing their novel ideas and knowledge with me. I also want to thank the rest of my thesis committee for their valuable comments.

I would like to extend my heartfelt thanks to my friends Upma Gandhi and Ali Farshidi, for their invaluable support and companionship throughout the course of my research. Also, their insights and encouragement have been a constant source of motivation and inspiration. Also, I would like to thank my fellow lab-mates Aysa, Andre, and Nima for their help and inspiring discussions in the lab.

Lastly, I would like to express my deepest gratitude to my parents for their support throughout my life. I am forever grateful for the sacrifices they have made on my behalf. I would also like to thank my sister and brother for their spiritual support. Lastly, I would like to thank my beloved partner Shadi for her love, encouragement, and tolerance, who made all the difference in my life.

Dedicated to all brave women and men who fight for their freedom. Women, Life, Freedom!

Table of Contents

List of Figures

List of Tables

List of Symbols, Abbreviations, and Nomenclature

Acronyms	**Definition**
$2D$	2-Dimension
$3D$	3-Dimension
ACM	Association for Computing Machinery
$ASP-DAC$	Asia and South Pacific Design Automation Conference
BFS	Breadth-First Search
BL	Baseline
CCG	Construct Constraint Graph
$CM-TCM$	Congestion aware Moving-Toward Cell's Median
CPU	Central processing unit
CRP	Cooperation between Routing and Placement
DAC	Design Automation Conference
$DATE$	Design Automation and Test in Europe
DBU	database unit
DEF	Design Exchange Format
DP	Detailed Placement
DPT	Dynamic Programming Technique
DR	Detailed Routing
DRC	Design Rule Check
DSP	Digital Signal Processing
$DTMF$	Dual Tone Multi-Frequency
ECC	Estimate Candidate Cost

EDA	Electronic Design Automation
FBC	Find Best Candidates
FM	Fiduccia–Mattheyses
FPM	Fixed Placement Mode
$G-Cell$	Global Routing Grid Cel
GB	Giga Byte
GCP	Generate Candidate Positions
GHz	Giga Hertz
GND	Ground nets
GR	Global Routing
HDL	Hardware-Description Language
$HPWL$	Half-Perimeter WireLength
$I/OPads$	Input/Output Circuits
IC	Integrated Circuit
$ICCAD$	International Conference on Computer-Aided Design
$IEEE$	Institue of Electrical and Electronic Engineers
ILP	Integer Linear Programming
$ILP-DP$	Integer Linear Programming-based Detailed Placement
$ISPD$	International Symposium on Physical Design
$ITRS$	International Technology Roadmap for Semiconductors
KL	Kernighan–Lin
LCC	Label Candidate Cells
LEF	Library Exchange Format
LVS	Layout Versus Schematic Checking
MBR	Minimum Bounding Rectangle
MST	Minimum Spanning Tree
MVC	Minimum Via Cost
$MWIS$	Maximum Weight Independent Set
$OFGV$	Out-of-Guide VIA
$OFGW$	Out-of-Guide Wiring
$OFTV$	Out-of-Track VIA
$OFTW$	Out-of-Track Wire

PC	Personal Computer
PCI	Peripheral Component Interconnect
PD	Placement Density
PR	Placement and Routing
RAM	Random Access Memories
RPM	Rip-up Placement Mode
$RSMT$	Rectilinear Steiner Minimum Tree
RTL	Register-Transfer Level
R^n	n-dimension real number
$S - ILP$	Solving ILP
$SI2$	Silicon Integration Initiative
$TCAD$	Transactions on Computer-Aided Design of Integrated Circuits and Systems
$TODAES$	Transactions on Design Automation of Electronic Systems
$TSMC$	Taiwan Semiconductor Manufacturing Company
UD	Update Database
VDD	Drain power voltage net
VIA	Vertical Interconnect Accessess
$VLSI$	Very Large Scale Integration
VSC	Via Cost Stack
WL	wirelength
WWW	Wrong Way Wiring
db	database
nm	nanometer
um	micrometer

Functions

C(e)	capacity an edge e
D(e)	demand an edge e
H(c)	Height of cell c
H(row)	Height of a circuit's row
L(e)	returns the layer to which edge e is assigned
MD(u, v)	Manhattan distance between the center of G-Cells u and v
$U_f(e)$	fixed usage of edge e

$U_w(e)$	wire usage of edge e
W(Site)	Width of circuit's site
W(c)	Width of cell c
X(left)	left-most x coordinate of the circuit
X(right)	right-most x coordinate of the circuit
X(c,p)	x coordinate shows the left edge of cell c in placement p
Y(bottom)	Lower coordination of the circuit
Y(up)	upper coordination of the circuit
Y(c,p)	y coordinate shows the bottom edge of cell c in placement p
$\delta(e)$	a number of VIAs involved in an edge e.
γ_{via}	represents a unit VIA multiplier to weight each VIA insertion
classifyNet(n)	A class of nets that shows the required algorithm for routing the net, you can create two class options: PATTERNROUTE and AStar. Within each class, you can define the specific algorithm that should be used for routing the net.
cost(c,p)	cost cell c in the placement p
cost(e)	cost an edge e
cost(path)	cost a path
cost(via)	cost a via
cost(wire)	cost a wire
$\text{costUnit}_{short}(L(e))$	shows the cost of short violation per DBU in the layer that edge is assigned to
$\text{costUnit}^{raw}_{short}(L(e))$	shows the short cost per unit area
$\text{costUnit}_{via}(u \rightarrow u')$	the unit cost of VIA insertion for G-Cell u to u' that are in adjacent layers
$\text{costUnit}_{wire}(L(e))$	shows the cost of wiring per DBU in the layer that edge is assigned to
$\text{cost}_{\text{short}}(e)$	a function that returns the cost of short violation for each edge.
$\text{cost}_c(i,j)$	A cost a cell in site i and row j.
costshort(L)	cost short violation in layer L
e(u,v)	an edge that connects G-Cell u to v
$f_i(x)$	a constraint function for a decision vector x
$f_o(x)$	an objective function for a decision vector x
len(C)	returns the length or size of the list C.
penalty(e(u, v)	penalty edge e that connect G-cell u to v
pitch(L(e))	returns the pitch of the layer to which edge e is assigned
pitch(L)	the distance between tracks in layer L

$\text{short}_{length}(e)$	represents the length of a short violation for edge e
via(u)	number of vias in G-Cell u
Sub-scripts	
i	index of a cell, node, edge, G-Cell, or iteration
idx	An identification for each object.
j	index of a cell, node, edge, G-Cell, or iteration
Scalars	
N_{row}	Number of row available in a window
N_{site}	Number of sites available in a window
S	Slope in the logistic function
α	it is a natural number including zero
β	it is a constant value used for estimating the number of edges in an edge
γ	The percentage of cells want to be moved.
θ	it is a natural number including zero
Sets	
C	Set of Cells
C_n	A list of cells connected to net n
E	Set of edges in a graph
E_{wire}	set of edges that incorporates a wire
L	Set of layers
N	Set of Nets
P_c	Set of placements for cell c
V	Set of vertices in a graph
candidate_{set}	a list of candidate cells to move
conflict_{cells}	A list of cells that are in conflict with a cell.
connected_{cells}	A list of cells that are connected by nets to a cell
movable_{set}	a list of movable cells with multiple candidate positions
placement_candidates	A list of coordinates in the form of (x, y) pairs.
Set Members	
b_i	a boundary for each constraint
c	a cell in a list of cells C
c_i	a coefficient

hist_c	a binary value indicates whether a cell c is selected for a candidate for moving
hist_m	a binary value indicates whether a cell c is moved or not
l	a layer in a list of layers
n	a net in a list of nets N
p	a placement
x_i	an integer variable in a decision vector
$\text{y}_c^{(i,j)}$	a binary value indicates whether a located in site i and row j.
y_c^p	a binary value indicates whether a placement p is selected for cell c

Vectors

weights	A list of cell index and candidate positions.
x	a decision vector

Epigraph

I didn't fail 1,000 times. The light bulb was an invention with 1,000 steps.

- Thomas Alva Edison

Chapter 1

Introduction

The birth of the Integrated Circuit (IC) revolutionized electronic devices and the lives of human beings. In 1949, when German engineer Werner Jacobi patented the idea of integrating electronic circuits into a single device, he lined up five transistors and used them as an amplifier [2]. The result as Jacobi recognized was the ability to shrink devices and make them cheaper to produce. In 1958, Jack Kilby assembled the first prototype of an IC from discrete electronic components and implemented it on a single chip. His prototype was primitive by today's standards and became the IC that we know today [3]. The world's first IC is shown in Figure 1.1. Although this figure doesn't look like much, this IC changed the world forever. Since then, we (as intelligent creatures) wasted no time in taking advantage of the IC's invention.

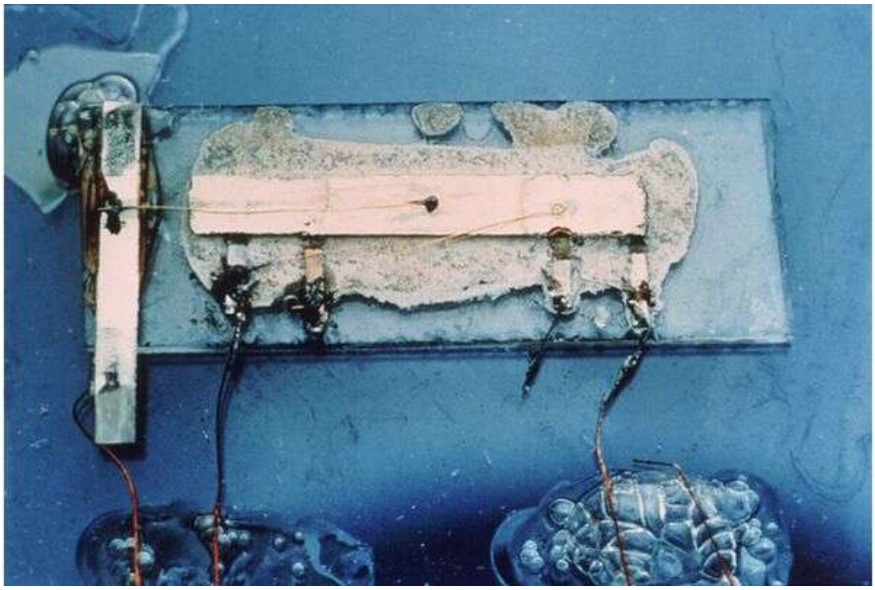
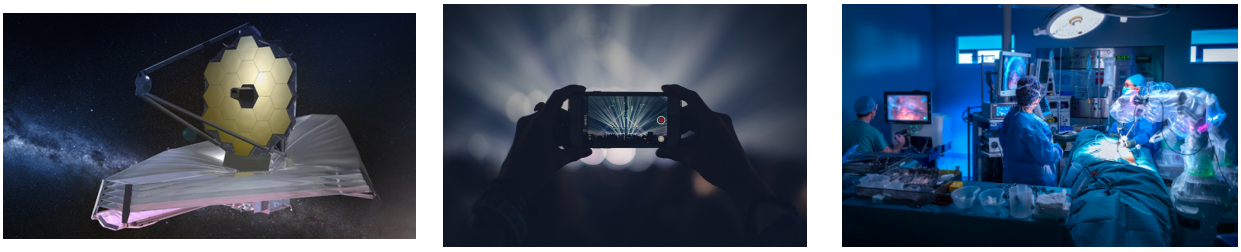


Figure 1.1: The world's first Integrated Circuit, Jack Kilby 1958 [3].

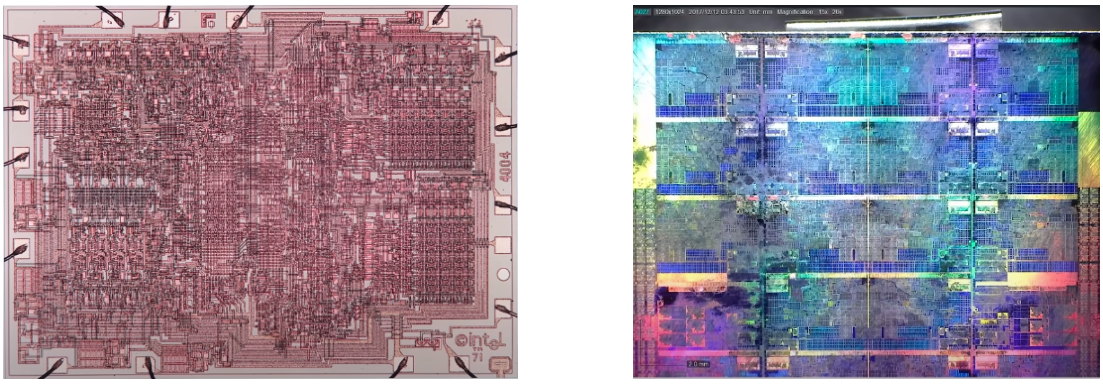
The advent of ICs in the last 60 years has changed the way we explore the universe (from ocean to space) and the way we find answers to our questions. Moreover, thanks to ICs, technology is available to society and impacts all aspects of our lives from the way we communicate to the way we entertain ourselves. Today, ICs are used in mobile phones, computers, and many other applications (Figure 1.2). Therefore, the importance of ICs and advancing that is clear. It helps us to improve the quality of our lives and reminds us how special human intelligence is.



(a) Telescope (b) Mobile Phones (c) Medical Devices

Figure 1.2: The usage of ICs in a wide range of applications [4].

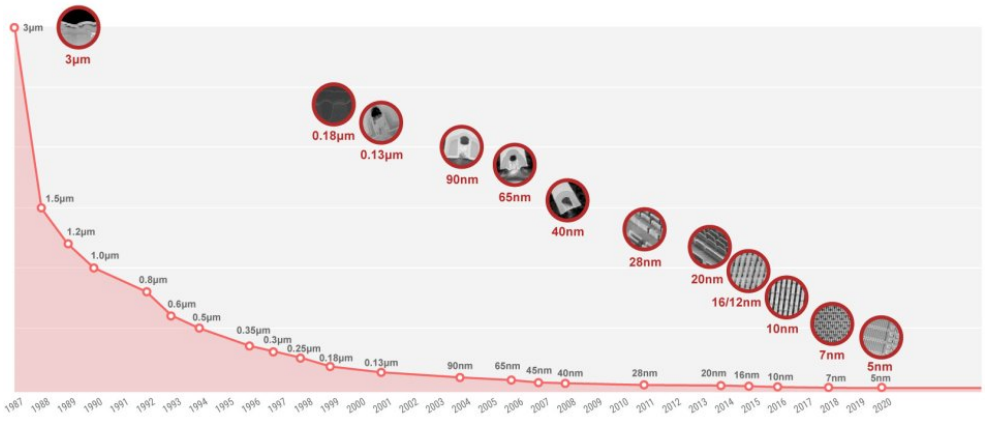
The process that combines hundreds of thousands, millions, and even billions of transistors in a chip is called Very Large Scale Integrated (VLSI) circuits. The advancement of the VLSI circuits has enabled designers to continuously increase the number of transistors in a chip. By now, it is possible to fabricate chips with millions or even billions of transistors. In a lifetime, we advanced from simple ICs with a few thousand transistors (e.g. Intel 4004 with 2300 transistors) Figure 1.3(a) to high-end chips (e.g. Intel i9-7980XE) consisting of billions of transistors Figure 1.3(b).



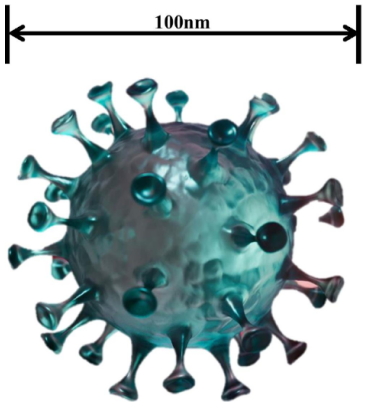
(a) Intel 4004: 2300 transistors 1971 (b) Core i9-7980XE: 7 billion transistors 2017

Figure 1.3: IC Advancement. (a) A microscopic image of Intel 4004 was released in 1971 [5] and around 2300 transistors are used for manufacturing by Intel company. (b) A nanometer image of Intel Core i9-7980XE with 7 billion transistors was released in 2017 [6]. Since the structures inside of IC have nanometer-sized and much smaller than the wavelength of light, the colors are not real.

This ever-increasing transistor density is made by shrinking the feature size of transistors and new generations of circuits. As transistor features change new design rules are implied to the manufacturing process of ICs which is called technology node. A technology node is a measurement of the smallest feature size that can be created on a semiconductor chip using a specific manufacturing process. It is typically used to refer to the size of transistors on a chip and is measured in nanometers (nm). The smaller the technology node, the more transistors can be packed into a given area, resulting in faster, more powerful, and more energy-efficient chips. The technology node is a key metric in the semiconductor industry, as it drives the development and advancement of new semiconductor devices. The advancement of the technology node over the last 30 years from 3um to 5nm is shown in Figure 1.4(a). It is worth mentioning that this nm technology node is smaller than the vast world of viruses. As you can see in the Figure 1.4(b), the diameter of COVID-19 viruses is around 100 nm [7].



(a) The advancement of technology node from 1987 to 2020



(b) The diameter of SARS-Cov-2 (COVID-19)

Figure 1.4: Technology Node Advancement. (a) The technology advancement reported by Taiwan Semiconductor Manufacturing Company (TSMC) [8] (b) The approximate diameter of COVID-19 [7].

In the early 1970s, we began to integrate thousands of transistors into a single chip. At the time, semicon-

ductor companies developed simple and customized software programs (called Electronic Design Automation (EDA)) to address their specific design styles. In the late 1980s, independent software companies created new software tools that provided broad use to meet designers' needs in IC design. For example, Synopsys [9] and Cadence [10] were two EDA companies that were founded in 1986 and 1988, respectively. Since then, EDA became a billion-dollar industry and each year several annual conferences present the progress of EDA in industry and academia. The EDA conferences and journals, I can name Design Automation Conference (DAC), International Conference on Computer-Aided Design (ICCAD), Design Automation and Test in Europe (DATE), Asia and South Pacific Design Automation Conference (ASP-DAC), Institue of Electrical and Electronic Engineers (IEEE) Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), and the Association for Computing Machinery (ACM) Transactions on Design Automation of Electronic Systems (TODAES).

The EDA tools were invented to automate the entire IC manufacturing process from design to validation. However, linking the design steps into a complete flow is challenging. Such integration is challenging because some design steps require tackling independently due to scalability challenges. The complexity imposed by a variety of chip designs and the continued increase in the number of transistors (or transistor density) makes the problem even more challenging. As the complexity of challenges grows and the level of integration increases, the cost of handling these issues by the EDA tools also grows. Therefore, cost-feasible IC products require innovation in EDA technology. This is demonstrated in Figure 1.5 that was released by the International Technology Roadmap for Semiconductors (ITRS), where the impact of EDA on the total IC design cost is presented. For example, according to this figure, in 2022, a cost of a chip design from Hardware Engineering, as well as EDA tool, remains manageable at $23.1 million. This cost will increase to $52.5 million with software design costs. Then, compare this value with 2012 where the total cost of IC design was estimated at around $105.3 million. Moreover, the cost of EDA has dropped significantly in 2013 and 2021. Although it is not easy to find the main reason for these drops, advances in technology can be a potential factor that could have contributed to the cost drop. it's possible that EDA tool vendors are able to develop more efficient and effective tools, which can reduce the overall cost of chip design. For example, improvements in machine learning algorithms could lead to faster and more accurate simulations, which could reduce the need for expensive hardware. Another reason can be Design Technology (DT) improvement. For example, in 2013, Concurrent software compiler enabled chip and Electronic System Design and Verification Enabled compilation and SW development in highly parallel processing SoCs [11]. This reduction in cost was achieved thanks to EDA technology innovation. This shows the importance of EDA technology, and the necessity of innovation to improve EDA tools to reduce overall IC design costs.

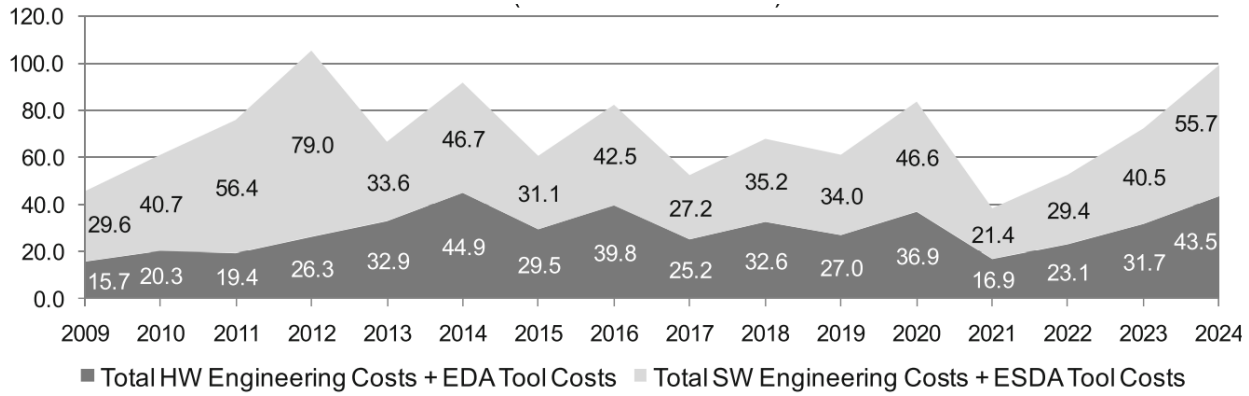


Figure 1.5: Technology roadmap for the cost of an IC equals the total hardware (HW) engineering cost + EDA tools cost (gray dark) and total software (SW) engineering costs + Electronic System Design Automation (ESDA) tool costs (light gray) [1].

1.1 Chip Design Flow

The chip design flow (or IC design flow) incorporates several steps [1] from system specification to chip fabrication, shown in Figure 1.6. It starts with *System Specification* where IC designers define the functionality and performance requirements of an IC including the ICs' input(s) and output(s). The basic architecture which satisfies IC requirements is determined in the *Architectural Design* step. It can include decisions about memory management techniques, types of computational cores, and standard external communication protocols. During the *Functional and Logic Design* step, the Register-Transfer Level (RTL) description of the system is described by a Hardware-Description Language (HDL). Then, the behavioral and modular description of the system is synthesized into a gate-level list that includes logical gates and their interconnections in the *Circuit Design* step. At this point, the design flow uses a pre-characterized cell library describing the logical, physical, and electrical characteristics of all the combinational and sequential elements available to the designer. This library is called the standard cell library.

During the *Physical Design*, the gate-level description (or logic description) of the circuit is converted into a geometric description called the layout. In other words, the physical design step maps the logic components to physical components. In this step, all cells are placed in the chip area and they are connected by net segments. After the circuit has been placed and routed (defined as placement and routing), the layout is checked during the *Physical Verification* step from different aspects to ensure that the system works correctly. Two main verifications can be named: Design Rule Check (DRC) and Layout Versus Schematic Checking (LVS). DRC verifies that the layout will satisfy technology file constraints, and LVS checks that the layout implements the exact schematic of the system. If an IC passes the physical verification, it will go to fabrication and packaging. Failure in each step of an IC design flow means failure in the final chip

manufacturing. The physical design step is one of the most challenging phases of automating the circuit design flow. The work presented in this thesis is focused on an improvement in the physical design flow, therefore, this step will be discussed in detail.

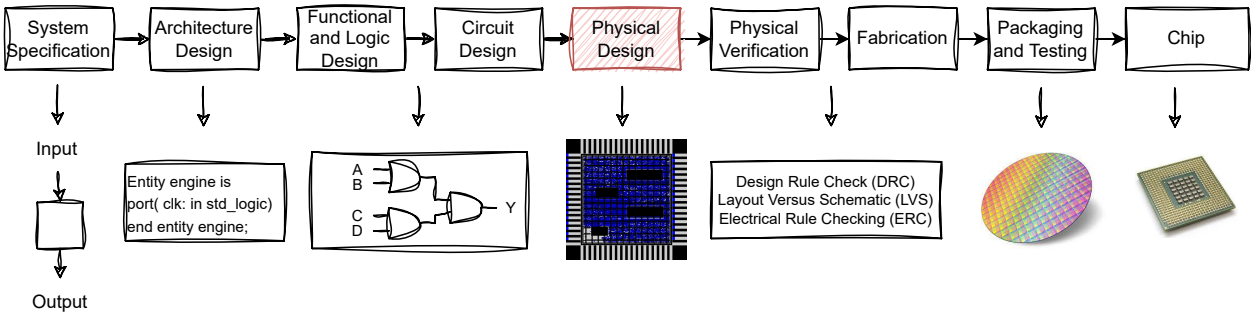


Figure 1.6: Circuit Design Flow [1].

1.2 Physical Design Flow

The main objective of the physical design flow is to determine the locations and connections between design components. In this step, the design rules (or design constraints) are required to be satisfied to guarantee a successful fabrication process of an IC. The input of the physical design is a gate-level design description and design rule's file (Figure 1.7(a)). The output of the physical design flow is the location and interconnection of the design components that satisfy design constraints (Figure 1.7(b)).

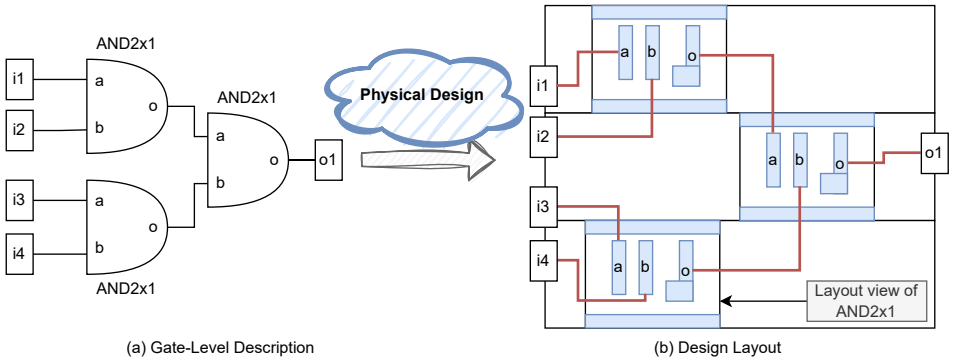


Figure 1.7: The physical design step converts a gate-level description of a circuit to a layout view or geometric description of the circuit.

With the increase in the circuit's size, and with new intricate design rules imposed by the technology nodes, the conversion from gate-level to geometric description becomes a complex and challenging task. Therefore, due to this complexity, the physical design flow is divided into six main steps (shown in Figure 1.8):

Figure 1.8: Physical Design Flow

- **Partitioning:** In this step, the circuit can be modeled as a graph. The main object of partitioning is to divide graphs into sub-graphs such that the number of connections between sub-graphs(sub-circuits) is minimized. This is shown in Figure 1.9 where two different cuts result in 4 and 2 external connections between two portions, respectively. Common partitioning techniques include Kernighan–Lin (KL) [12] and hMETIS [13] as a few partitioning techniques can name.

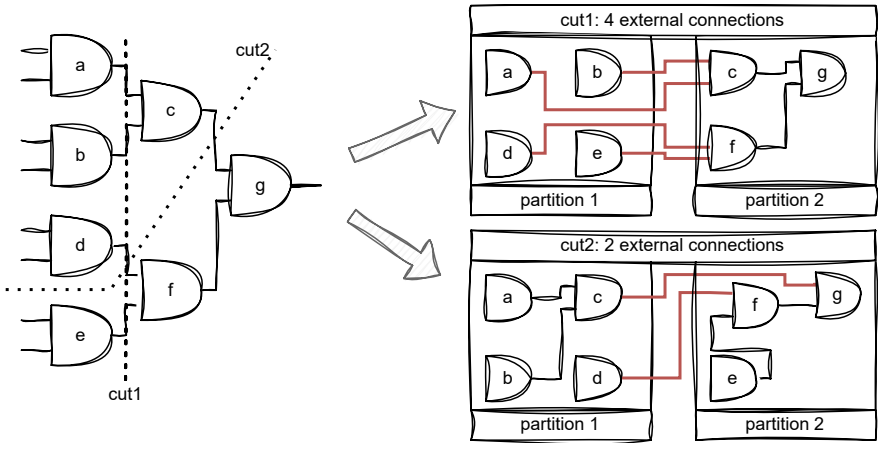


Figure 1.9: Two different partitioning results in 4 and 2 external connections. The main objective of partition is to minimize the external connection between divided blocks (or partitions).

- **Chip Planning:** The gate-level description of a circuit can be partitioned into modules. As shown in Figure 1.10, The modules become blocks when their shape and location are assigned in the layout. In chip planning, the shapes and locations of the sub-circuits are determined to satisfy some metrics such as the minimum area of layout and wire length. Cluster growth [14] and simulated annealing [15] are two common algorithms in the chip Planning step. Also, in recent works, using a deep reinforcement learning approach [16, 17] showed a potential improvement in chip planning [18, 19].

- **Placement:** The placement, in this thesis, refers to standard cell placement. The standard cell means that for each logical gate, a predefined block that has a fixed size and functionality is used during placement. A sample of standard cell designs for NAND logic is given in Appendix A. During placement (or standard cell placement), the locations of all the standard cells in the circuit are determined. Due to complexity, the placement step is divided into three steps: Global Placement [20], Legalization [21–23], and Detailed placement. The steps of placement are shown in Figure 1.11. During global placement,

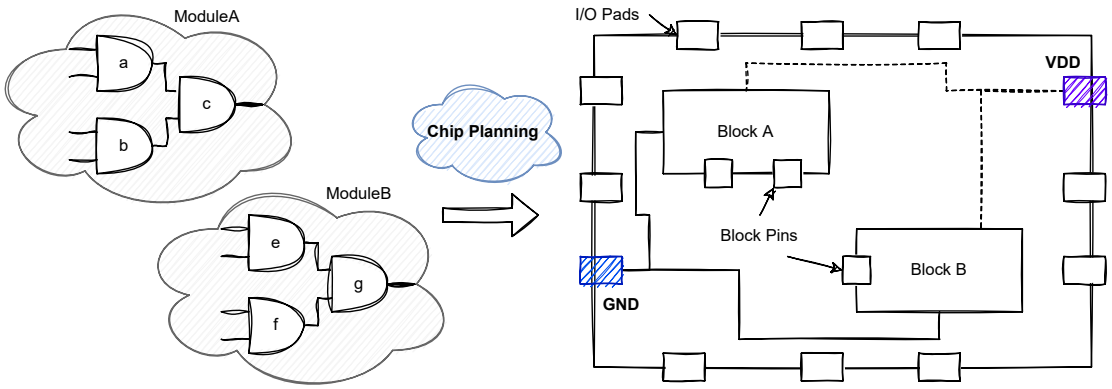


Figure 1.10: Chip planning [1].

an estimate of the location of cells is determined Figure 1.11(a). In this step, the overlap between cells (light blue rectangles in Figure 1.11(a)) is allowed. Then, during the legalization, one of the main tasks is to remove the overlap between components while displacement from initial placement is minimized (Figure 1.11(b)). Finally, during the detailed placement step, further improvements are applied to the placement solution.

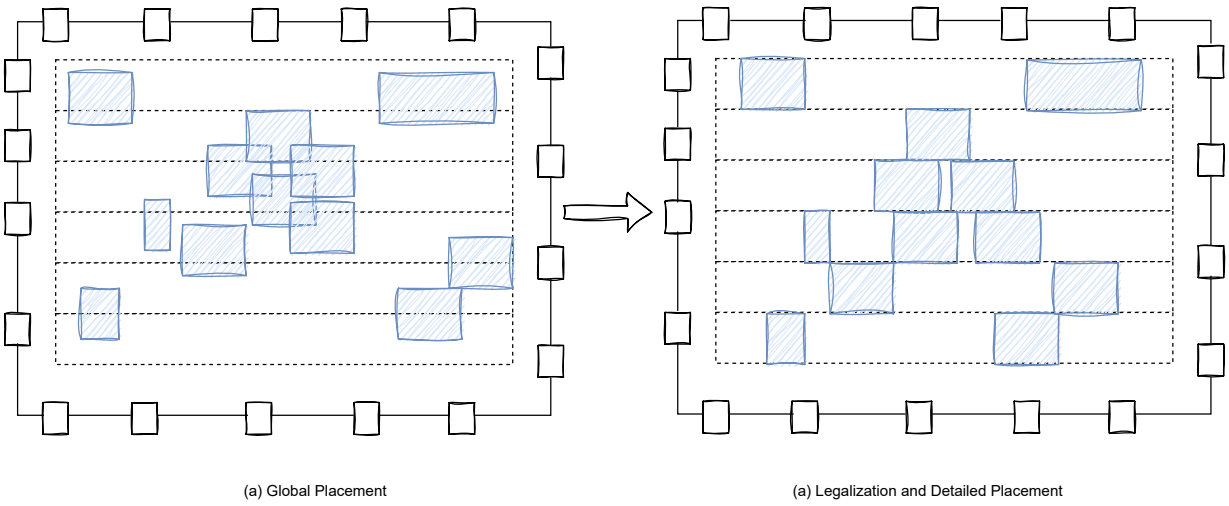


Figure 1.11: Standard Cell Placement Steps

- **Routing:** Due to the complexity of the routing step, it is divided into global and detailed routing [1]. The routing resources are modeled during global routing by coarse routing grids called G-Cells [24, 25]. The main objective of global routing is to determine routing regions that each net will be routed through Figure 1.12(a). The global routing solution provides an approximation of a routing solution. The detailed routing step is performed based on the global routing solution with the objective of finding the exact routes of each net (Figure 1.12(b)). Since the detailed routing step is performed according

to the global routing solution, the quality of the routing solution is highly dependent on the global routing solution [26].

During the detailed routing step, the routing resources are modeled as fine-grain. The space search and complexity of routing an individual net in the detailed routing step is much larger than in the global routing step. The resource modeling in global routing is simplified and it has less complexity compared to the detailed routing step [27]. In two ways global routing helps the detailed routing. First, the detailed routing confines its search space for each net to the routing regions determined by the global routing. This can reduce the search space during the detailed routing step. Second, due to the high problem complexity in the detailed routing step, it is hard to apply sophisticated algorithms in this step. Therefore, the global routing addresses several metrics such as routability, wirelength, etc to be optimized for all nets [24, 25, 28, 29].

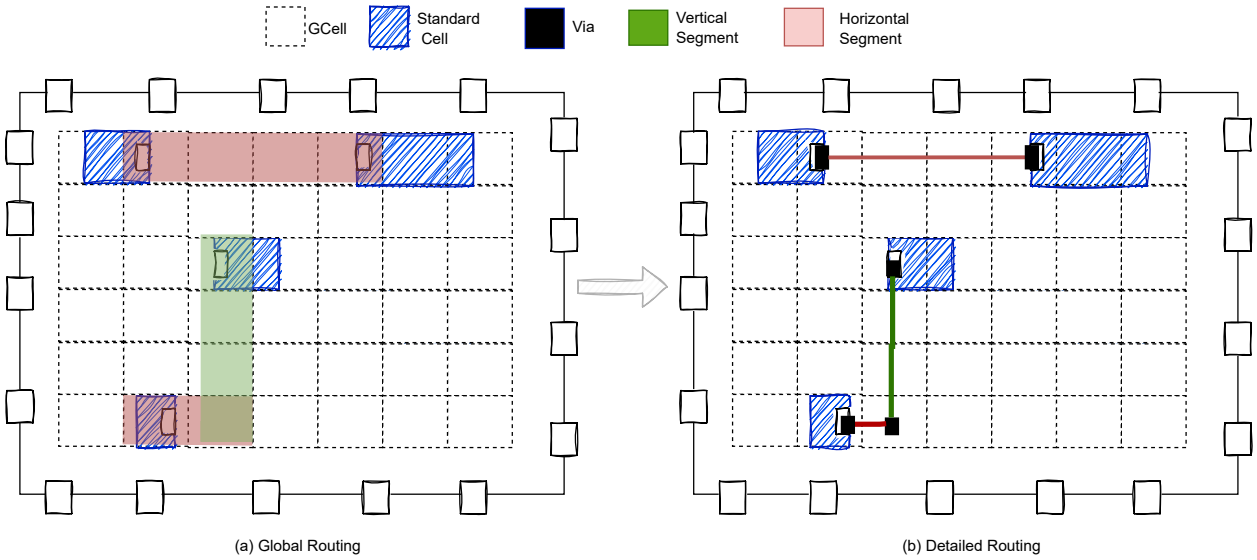


Figure 1.12: Global and Detailed Routing

- **Timing Clousre:** The process that meets the timing requirements of an IC design is called timing closure. The components of timing closure can be divided into Timing-driven placement, Timing-driven routing, and physical synthesis improvement by changing the netlist [1]. The main goal of Timing-driven placement is to minimize signal delays in a circuit by changing the location of the circuit's components [30]. During Timing-driven routing also the signal delays can be optimized by changing routing topologies [31]. Finally, the physical synthesis can improve the timing by changing the netlist. For example, a common approach is to insert buffers into nets to decrease their propagation delays [32].

1.3 Literature Gaps

The Placement and Routing (PR) are two key steps of the physical design flow that can impact the circuit performance, power consumption, and the manufacturing feasibility of the IC design [33]. Due to the complexity of development, the placement and routing are developed as independent steps, and traditionally they are solved separately [27].

However, the continued shrinking in the size of transistors and increase in the transistor density has diminished the boundaries and the abstraction that separate these two design steps. For example, multi-layer interconnection issues such as 3D routing congestion are required to be accurately calculated during the placement. If 3D routing issues can not be captured during the placement, this can result in a placement being deemed unroutable and the whole process being repeated. This means a significant increase in design time and costs. Therefore, the placement and routing of the physical design process are moving from a sequence of independent steps toward a deeper level of integration. But, such integration is challenging. The integration between placement and routing has become an important problem in EDA and it has been the subject of two recent contests: 2020 and 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD) [34, 35].

Although the ICCAD contests brought up the importance of studying the idea of integration (or cooperation) between routing and placement, the benchmarks provided during the ICCAD contests were far away from realistic benchmarks with technology design rules. The ICCAD benchmarks had no detailed routing information and there was no data regarding actual wiring, nor complex detailed routing rules. All benchmarks proposed during the ICCAD contests were considerably simplified with small circuit sizes that had low placement density. Therefore, it has not been clarified the impact of the techniques proposed during the contests on the detailed routing solution, especially when the circuit has high placement density. Detailed routing is a significantly complex and time-consuming step. Considering the detailed routing solution quality will highly impact the development complexity of integration between routing and placement. Thus, there is a lack of studies on the integration of routing and placement and investigation of its effect on the detailed routing solution in circuits with high placement density.

1.4 Motivations and Research Hypothesis

A placement engine is used to determine the location of standard cells and evaluate the routability of a placement solution. Then, conventionally, the location of standard cells is fixed, and a router engine is used to connect the cells. A half-perimeter wirelength (HPWL) model is commonly used to estimate wirelength

during placement [22, 36, 37]. Also, standard cell density is used to model routability during placement, which is a necessity but not sufficient to ensure routability [34]. Because as the complexity of circuits grows, such as placement density and new intricate design rules, there is a possibility the objectives and constraints defined during placement are not well correlated with the routing step. One solution to mitigate this problem is to build cooperation between routing and placement [33]. There are a few works that used global routers to estimate the routability of a placement solution during placement [38–41]. However, invoking routing several times to consult with the placement can be highly runtime-consuming. Moreover, none of these works reported the effect of their techniques on the detailed routing solution. In 2018 and 2019, for the first time, the EDA industry released benchmarks with detailed routing information during ISPD 2018 and 2019 [42,43]. Since then, there has not been any academic work that applies placement and routing to these benchmarks. Therefore, the lack of a framework that cooperates with routing and placement and reported the results after the detailed routing solution rises, and it is important to be addressed. The motivation of this thesis is to develop a framework that implements efficient cooperation between global routing and detailed placement to improve the detailed routing solution.

The research hypotheses of this thesis are:

- **Hypothesis 1:** A framework that builds cooperation between global routing and detailed placement to improve the detailed routing solution can be developed by (1) identifying standard cells that require to be moved (2) generating new placement for selected standard cells (3) the new placement required to be legalized (4) for each placement related global routing solution generated (5) choosing the best global routing solution and associated placement. Improvement in the global routing solution can improve the detailed routing solution if the global routing is well correlated with the detailed routing.

- **Hypothesis 2:** A detailed placement algorithm that moves standard cells to their optimal region can provide high-quality placement candidates to reduce the frequency of invoking global routing. This can improve the runtime of cooperation between global routing and detailed placement.

- **Hypothesis 3:** Storing expensive functions in memory and calling back from there if they are required again can reduce the runtime of global routing by avoiding recalculation.

1.5 Contributions

In this thesis, a framework that incorporates cooperation between routing and placement with the existence of technology node constraints is proposed. The main objective of the proposed framework is to improve the

detailed routing score by combining routing and placement. The core of this framework is the CRP engine. The following major contributions are presented in this thesis:

- Proposing and developing a framework that combines global routing with detailed placement to improve detailed routing score.
- Proposing an efficient cooperation engine between routing and placement called CRP. This engine can be used to fill the gap between the placement and routing steps to empower the routing solution.
- Developing an ILP-based Detailed Placement (ILP-DP) with a new cost function called Congestion-aware Moving-Toward Cell's Median (CM-TCM) to generate different legalized placement candidates for each potential candidate cell to move.
- Proposing two modes in global routing: Fixed Placement Mode (FPM) and Rip-up Placement Mode (RPM). To reduce the runtime of global routing, a caching technique (called Cost Caching) is proposed to accelerate the cost estimation by using global routing. This technique can reduce the runtime of global routing by 28.56% compared with the state-of-the-art. In addition, a Net Caching technique is proposed to execute the cooperation between routing and placement in parallel. This results in a speedup of 2.6 times compared to a sequential approach.
- Demonstrating the effectiveness of the proposed techniques on benchmarks that include detailed routing information such as ISPD 2018 and ISPD 2019. Evaluating the results in these benchmark suites enables us to investigate the quality of proposed techniques on the detailed routing metrics.

The contributions of this thesis were published as research papers in [44, 45].

1.6 Thesis Structure

This thesis consists of four chapters including this chapter. The rest of this thesis is structured as follows:

- **Chapter 2:** The physical design terminology is illustrated in the beginning. The problem of routing with cell movement is then formulated. A standard format for placement and routing description is presented next. The background of routing and placement methods are reviewed after that. The literature and gaps in the cooperation between routing and placement are then discussed, as the main focus of this thesis. Finally, the challenges in developing a framework for cooperation between routing and placement are discussed.

- **Chapter 3:** This chapter presents the concepts that form the foundation of the methods developed in this thesis to address placement and routing challenges discussed in chapter 4. The chapter is organized as follows: the main concepts for complexity analysis are provided at the beginning. The preliminary concepts of graph theory used in this thesis are defined next. After that, the main data structure used to organize and handle the query process of the layout is discussed. Next, The sequence of steps and main properties of the dynamic programming technique are discussed, followed by a discussion of the ILP model for solving optimization problems. Finally, The heuristic methods and various approaches within heuristic methods are explained and illustrated.

- **Chapter 4:** The proposed framework is outlined in this chapter. First, the cost functions used in the Cooperation between Routing and Placement (CRP) engine are defined. Next, the five main steps of the CRP engine are described in detail. Lastly, an illustration of the developed Integer Linear Programming with Detailed Placement (ILP-DP) and extended global routing methods are provided.

- **Chapter 5:** In this chapter, the experimental results are presented. First, the experimental setup is outlined. Then, the characteristics of the two benchmark sets used in the experiments, the ISPD 2018 [42] and ISPD 2019 [43] contests, are discussed. The evaluation process and metrics are then described, followed by the experimental results for each benchmark. The techniques and methods used to improve the state-of-the-art are analyzed, and finally, the performance of the Cooperation between Routing and Placement (CRP) engine in terms of runtime and memory usage is reported.

- **Chapter 6:** In this chapter, concluding results and recommendations, as well as suggestions for future research, are given.

Chapter 2

Background and Related Work

2.1 Introduction

The framework introduced in this thesis is developed to combine routing and placement steps in the physical design flow. The concepts presented in this chapter are the technical terms and related works of the placement, routing, and cooperation between routing and placement problems.

The rest of this chapter is organized as follows: The physical design terminology is illustrated in Section 2.2. The problem of routing with cell movement is then formulated in Section 2.3. A standard format for placement and routing description is presented next (Section 2.4). The background of routing and placement methods are reviewed after that in Section 2.5. The literature and gaps in the cooperation between routing and placement are then discussed, as the main focus of this thesis Section 2.6. Finally, the challenges in developing a framework for cooperation between routing and placement are discussed in Section 2.7.

2.2 Physical Design Terminology

In this section, technical terms used during the placement and routing steps of physical design are illustrated. A standard cell is a method of designing Application-Specific Integrated Circuits (ASICs). A top view of a circuit using a standard cell is shown in Figure 2.1. The total partition area of a circuit, in the standard cell design, can be defined as a rectangle called a *die area*. A smaller rectangle inside the die area is called a core area, where components in a circuit are placed. These components are called *macros*. Each component can represent a logical gate like NAND, NOR, XOR, etc. A component also can represent a memory block like Flip-flops or Random Access Memories (RAM), etc. An instance of each component is called a *cell*. The cells are gray rectangles in Figure 2.1. The inputs and outputs of a cell are called *pins*. Pins are blue

rectangles inside each cell and they can have complex geometries. In a standard cell design, cells must not have any overlap and they must be aligned to specific rows. The *rows* are horizontal rectangles stacked vertically in the die area, as shown in Figure 2.1. Also, cells can only be placed in specific spaces that are called *sites*. The sites are small vertical rectangles that are sequentially placed in each row. A sample site that is highlighted in purple is displayed in Figure 2.1.

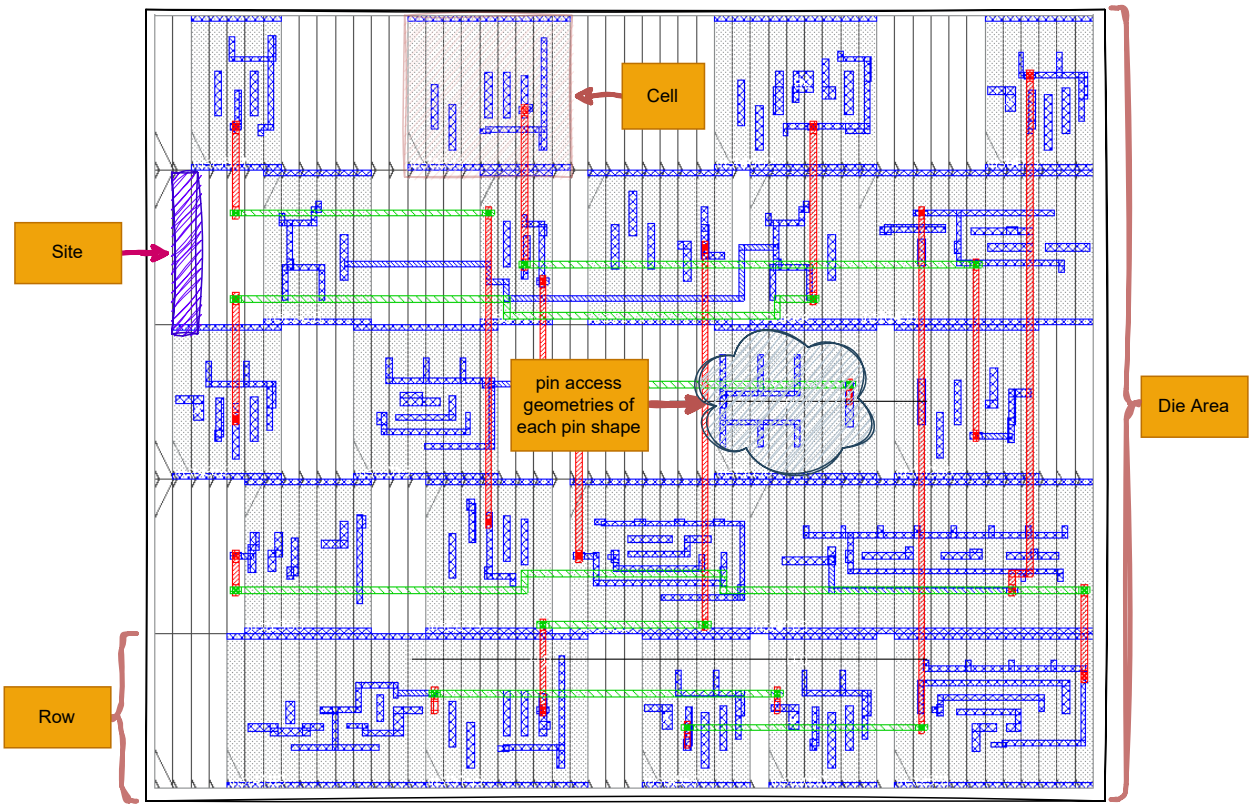


Figure 2.1: A top view of a circuit layout (this layout shows ispd18_sample benchmark).

The metal segments that connect all the cells that need to be connected are called a *net*. The combination of red, green, and purple segments in Figure 2.2 shows an example of a net that connects two cells. Each Net is made of one or multiple segments called *wire*, red and green segments in Figure 2.2. A wire only can pass through pre-defined paths called *tracks*. In each layer, the routing tracks run either horizontally or vertically. The tracks are shown in solid gray lines in Figure 2.2. The wires of two different layers can be connected by Vertical Interconnect Accesses (VIAs), purple segments in Figure 2.2. It is common to call the first layer a pin layer, and the layers above that are respectively cut and routing layers. The cut layer specifies the VIA insertion rules and routing layers are defining characteristics of wiring. A sample cut layer is shown between layer 1 and 2 routing layers in Figure 2.2.

In a standard cell design, the smallest possible size to describe a circuit is called a database unit (DBU).

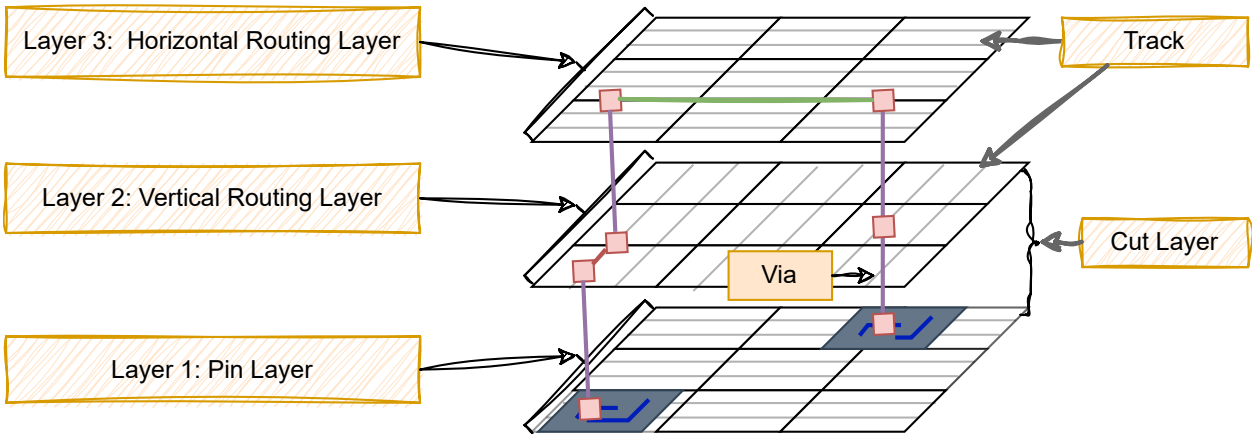


Figure 2.2: A 3D view of a circuit layout representing routing terminology.

This metric is used to ease scaling a design circuit into a real-world size. The description of key concepts used in this thesis is given in Table 2.1. For further information about the physical design terminology please refer to [46].

Table 2.1: Physical Design Terminology

#	Term	Description
1	*cell*	An instance of a macro.
2	*circuit*	An electrical and logical element that are connected by wires to each other is called a circuit.
3	*core area*	A partition of a circuit that components or cells are placed.
4	*die area*	The total partition area of a circuit is called the die area.
5	*macro*	Macro is an intellectual property such as logical gates or memory blocks.
6	*metal layer*	A manufacturing process level in which design components are placed.
7	*net*	A net is an interconnection between two or more pins that make them electrically equivalent.
8	*netlist*	The list of the nets that describes the circuit.
9	*pin*	A pin is defined as the input and output connection of macros.
10	*pitch*	The pitch size is the distance between horizontal and vertical tracks in each layer.
11	*row*	The area defined in a design for standard cell placement.
12	*site*	Legal locations that standard cells can be placed in a design.
13	*track*	Track is a legal path that nets can pass through.
14	*VIA*	VIA is a metal that makes a connection between layers.
15	*wire*	A wire is a metal that makes two points electrically equivalent.

2.3 Problem Formulation

This work combines global routing with detailed placement to improve the detailed routing solution. The input to the global routing is a placed and legalized circuit. In the global routing formulation, the area of the

circuit is partitioned into regions called G-Cells (Grid Cells), where the 3D routing space can be modeled as a 3D graph of G-Cells (G). Each edge of graph G is notated by e, and it has two attributes: demand ($D(e)$) and capacity ($C(e)$). The main objective is to find a placement for each cell that minimizes the total cost in Equation 2.1.

$$\min \sum_{c \in C} \sum_{p \in P_c} cost(c,p) \times y_c^p \tag{2.1}$$

In Equation 2.1, y_c^p is a binary value that indicates whether a placement p is selected for cell c. The $cost(c,p)$ represents the cost of p for cell c, and it defines the path cost of all routed nets connected to cell c (path cost for a net is explained in Definition 4.6). After each movement, the following constraints must be satisfied to have a valid and legalized placement solution.

Table 2.2: Symbols used in the problem formulation

Symbol	Definition	Symbol	Definition
C	Set of Cells	$W(Site)$	Width of circuit's site
c	A cell in a list of cells C	$X(c,p)$	x coordinate shows the left edge of cell c in placement p
$cost(c,p)$	cost placement (or position) p for cell c	X(left)	left-most x coordinate of the circuit
$H(c)$	Height of cell c	X(right)	right-most x coordinate of the circuit
$H(row)$	Height of circuit's row	Y(bottom)	Lower coordination of the circuit
P_c	Set of placements (or positions) for cell c	Y(up)	upper coordination of the circuit
p	A placement	$Y(c,p)$	y coordinate shows the bottom edge of cell c in placement p
$W(c)$	Width of cell c	y_c^p	y_c^p=1, then placement p is selected for cell c

$$\sum_{p \in P_c} y_c^p = 1, \ \forall c \in C \tag{2.2}$$

$$y_c^p \in \{0,1\} \tag{2.3}$$

For each cell $c \in C$, consider the placement p where $y_c^p = 1$, then:

$$X(\text{left}) \leq X(c,p) \leq X(\text{right}) - W(c), \ Y(\text{bottom}) \leq Y(c,p) \leq Y(\text{up}) - H(c) \tag{2.4}$$

$$\begin{cases} X(c,p) < X(c',p) \mapsto X(c,p) + W(c) \leq X(c',p) \\ X(c',p) < X(c,p) \mapsto X(c',p) + W(c) \leq X(c,p) \\ c, c^{'} \in C, c \neq c^{'}, \text{both } c \text{ and } c^{'} \text{ are in the same row, and } y_{c'}^p = 1 \end{cases} \tag{2.5}$$

$$X(c,p) = \alpha \times W(site), \alpha \in \mathbb{N} \tag{2.6}$$

$$Y(c,p) = \theta \times H(row), \theta \in \mathbb{N} \tag{2.7}$$

Equation 2.2 ensures that each cell must have only one selected position. In Equation 2.3, y_c^p either can be 1 or 0, if y_c^p=1, then placement p is selected for cell c. In Equation 2.4, it states that each position of a cell must be inside the circuit. In this equation, $X(left/right)$ and $Y(bottom/up)$ show the boundary and corners of the circuit, and $W(c)$ and $H(c)$ show the width and height of a cell c. After that, Equation 2.5 states that cells must not have any overlap to be considered as a legalized solution. In this equation, $X(c,p)$ shows the x coordinate[1] of a cell c in a placement p, and $Y(c,p)$ shows the y coordinate of the cell c in the placement p [2]. Both cells (c and c') are considered abutting cells in the same row. In Equation 2.6, it ensures that each cell is aligned to sites horizontally (α is a natural number including zero) [47]. Finally, Equation 2.7 states that each cell is aligned to power and ground rails on the top and bottom sides of each row (θ is a natural number including zero). Note: Each row has a specific orientation that all cells placed in the row must follow [47]. With the aforementioned constraints, I can ensure that the circuit is completely legalized. This means that the input is valid to apply to any detailed router.

2.4 Circuit Description by using LEF and DEF Format

Library Exchange Format (LEF) and Design Exchange Format (DEF) are standards for place and route design tools in EDA and they are developed by Cadence Design System and distributed by Silicon Integration Initiative (SI2) [10, 48]. The LEF file format includes the physical design characteristics and technology constraints of a circuit. A simplified version of a LEF file format is shown in Figure 2.2. In this figure, each keyword is highlighted in red and defines elements of an IC process technology. The constraints defined in the LEF file must be satisfied in the placement and routing solution in order to meet the manufacturing requirements of foundries. In this thesis the following information is used from LEF and DEF files to model routing and placement problems in the physical design step:

LEF File (shown in Figure 2.2): The LEF file is used to define the design rules of a physical layout of an IC. The LEF file can include the following:

- **UNIT:** Specifies the DBU per micron. This metric is used to convert all distance units into microns. For example, if the DBUperMicron in UNITS is equal to 2000, it means that 1 micron equals 2000 database units in the circuit's scale.

- **SITE:** Defines a placement site in a design. This can also include the class of SITE that can be Input/ Output Circuits (I/O Pads) sites and Core Sites for placement. In different technology nodes, the width and height of each SITE can be different.

[1] x coordinate shows the left corner of a cell.
[2] y coordinate shows the bottom corner of a cell.

```
UNITS
   DATABASE MICRONS [DBUperMicron]; // Specify the the Database Units (DBU) per micron (DBUperMicron)
                                    // DBUperMicron: to convert DEF distance units into microns
                                    //              Example: if DBUperMicron is equal to 2000 it means that 1 micron
                                    //              equals 2000 units in the circuit scale.
END UNITS

SITE [siteName]                     // Define a placement site in the design.
   CLASS [coreClass] ;              // coreClass: Specifies a class for the site. The site either can be an I/O PAD site
                                    // or Core Site
   SIZE [width] BY [height] ;       //Width and height: Width and height of a site's class
END CoreSite

LAYER Metal1
   TYPE ROUTING ;
   DIRECTION [HORIZONTAL | VERTICAL];// Specifies the routing direction
   MINWIDTH [minWidth] ;            // Specifies the minimum legal object width on the routing layer
   AREA [area] ;                    // Specifies the minimum metal area required for polygons on the layer
   WIDTH [width] ;                  // Specifies the default routing width to use for all regular wiring on the layer
   SPACING [spacing] ;              // Specifies the spacing rules to use for wiring on the layer. You can specify
                                    // more than one spacing rule for a layer.
   SPACING [eolSpace] ENDOFLINE [eolWidth] WITHIN [eolWidthin] ; // indicates that an edge that is shorter than eolwidth
                                                                 // noted as end-of-line (EOL) edge requires spacing
                                                                 // greater than or equal to eolSpace beyond the EOL
                                                                 // that is less than eolWidthin.
   SPACINGTABLE
   PARALLELRUNLENGTH
     WIDTH  [width1]  [spacing1]    // specifies the maximum parallel run length between two objects
     WIDTH  [width2]  [spacing2] ;  // if the maximum width of two objects are greater than width1, then
                                    // spacing between two objects must be spacing1.
   PITCH [X] [Y] ;                  // Specifies the required routing pitch for the layer. Pitch is used to generate
                                    // the routing grid (Tracks in DEF files).
END Metal1

LAYER Via1
   TYPE CUT ;
   SPACING [spacing] ;              // Specifies the minimum spacing allowed between the via cuts on the same net or
                                    // different nets.
   WIDTH [width] ;                  // Specifies the minimum width of a cut.
END Via1


MACRO [macroName]
   CLASS CORE;
   ORIGIN [X] [Y];                  // Specifies how to find the origin of the macro to align with a DEF component
                                      // placement point.
   SIZE [width] BY [height] ;       // Specifies a placement bounding rectangle for the macro.
   SYMMETRY X Y ;                   // Specifies which macro orientations should be attempted by the placer before
                                    // matching to the site. It means that all N, FN, FS, and S orientations should be tried.
   SITE CoreSite ;                  // specifies the site associated with the macro.
   PIN A                            // Define pin for the macro
      DIRECTION INPUT ;             // Specifies the pin type.
      USE SIGNAL ;                  // Specifies how the pin is used.
      PORT                          // defines a pin port statement that defines a collection of geometries
      LAYER [layerName] ;           // that are electrically equivalent points
      RECT [XL] [YL] [XH] [YH];     // A Rectangle that describe a port in the pin A.
      END
   END A
END [macroName]
```

Figure 2.2: Library Exchange Format (LEF) file.

- **LAYER:** Defines a manufacturing process level in which design components are placed. Two types of layers are used in this thesis: Routing Layer and Cut Layer.
  - **Routing Layer:** For each layer, the following information is specified:
    * DIRECTION: This metric specifies the preferred routing direction in each layer.
    * MINWIDTH: This metric specifies the minimum legal object width for each layer. It is not a valid object width If a metal object is smaller than MINWIDTH.
    * AREA: This metric specifies a minimum metal area required for each layer.
    * WIDTH: A default routing width to use for all regular wiring on each layer. According to the type of net, the width of each net can be different. There two types of nets are addressed in this work signal nets and special nets (VDD/GND). Special Nets (or Supply nets) use thick metal layers and are routed on top layers due to high current loads. Also, the wider wires have less voltage resistance and therefore have a lower voltage drop [27].
    * SPACING: This metric specifies spacing rules to use between wires in each layer.
    * PARALLELLENGTH: This metric specifies a maximum parallel run length between two objects in each layer.
    * PITCH: This metric specifies the routing grid tracks defined in the DEF file.
  - **CUT Layer:** The following metrics can be specified in each cut layer:
    * SPACING: This metric specify a minimum spacing allowed between VIAs in a cut layer.
    * WIDTH: This metric specifies a minimum width of an object in each cut layer.
- **MACRO:** Defines an intellectual property such as logical gates or memory blocks. The following metrics can be specified in the definition of each Macro:
  - ORIGIN: This metric specifies the origin of the macro to align with a DEF component in a placement point.
  - SIZE: This metric specifies the placement bounding rectangle.
  - SYMMETRY: This metric specifies the orientation of the macro.
  - SITE: This metric specifies a SITE associated with the macro.
  - PIN: This metric defines the input and output pin access geometries that can be used to access the macro.

DEF File (shown in Figure 2.3): The DEF file is used to represent a physical layout of an IC. The DEF file can include the following:

```
DESIGN [designName] ;                                  // Specifies a name for the design.

DIEAREA ( [XL] [YL] ) ( [XH] [YH] ) ;                  // Specifies two corners of the bounding rectangle for the design.


ROW [rowName] [siteName] [origX] [origY] [siteOrient]
        DO [numX] BY [numY] STEP [stepX] [stepY];      // define row in the design
                                                       // numX and numY: Specifies the repeating set of the sites to create the row.
                                                       // origX and origY: Specifies the location of the first site in the row.
                                                       // rowName: Specifies a name for the row
                                                       // siteName: Specifies the LEF site used for the row.
                                                       // siteOrient: Specifies the orientation of all sites in the row.
                                                       // stepX and stepY: Specifies the spacing between sites in horizontal and vertical
                                                       // rows.



TRACKS [X | Y] [start] DO [numTracks]
      STEP [space] LAYER [layerName] ;                 // Defines the routing grid for a standard cell-based design.
                                                       // X | Y start: Specifies the location and direction of the first track.
                                                       // numTracks: Specifies the number of tracks created for the grid.
                                                       // space: Specifies the spacing between tracks.
                                                       // layerName: Specifies the routing layer used for the tracks.


COMPONENTS [numComponents] ;                           // Defines design components, their location, and associated attributes.
- [compName] [macroName] + [Fixed | PLACED] ( [XL] [YL] ) [orient] ;        // compName: Specifies the component name.
                                                       // macroName: Specifies the component being defined should be
                                                       //          electrically equivalent to the defined macro in LEF.
                                                       // Fixed: Specifies the component has a location and cannot be moved.
                                                       // PLACED: Specifies the component has a location and can be moved.
                                                       // XL YL: Specifies the bottom-left corner location of the component.
                                                       // orient: Specifies the orienation of the component.
END COMPONENTS


NETS [numNets] ;                                       // Defines the netlist connectivity.
- [netName]                                            // Specifies a name of a net.
  ( [compName] [pinName] ) ( compName] [pinName] )     // Specifies the name of a regular component pin on a net.
 ;
END NETS


END DESIGN
```

Figure 2.3: Design Exchange Format (DEF) file.

- **DESIGN:** Specifies a name for the design.
- **DIEAREA:** Specifies two corners of the bounding rectangle for the design.
- **ROW:** The following information can be specified to define a row in the design.
  - rowName: This metric shows the name of a row.
  - siteName: This metric specifies the site name defined in the LEF file and associated with the row.
  - origX and origY: These metrics specify a location for the first site in the row.
  - siteOrientation: This metric specifies the orientation of all in the row.
  - numX and numY: This metric specifies the number of sites to create the row.

  - stepX and stepY: This metric specifies spacing between sites in horizontal and vertical rows.

- **TRACKS:** The following information can be specified to define a routing grid for a standard cell-based design. Tracks can are used as pre-defined paths for wiring.
  - X or Y start: This metric specifies a location for the first track.
  - numTracks: This metric specifies the number of tracks created for the routing grid.
  - space: This metric specifies the spacing between tracks.
  - layerName: This metric specifies the name of the layer associated with the track.

- **COMPONENTS:** The following metrics can be specified to define an instance of a component in the design.
  - compName: This metric specifies a name for the component.
  - macroName: This metric specifies the associated macro with the component.
  - FIXED or PLACED: This metric specifies the given location for the component. In the case, the given location is fixed, it means that the component is not movable otherwise it is movable.
  - XL or YL: This metric specifies a bottom-left corner location for the component.
  - orient: this metric specifies the component's orientation. There are eight different orientations each component can have such as zero rotation (R0), 90 °rotation (R90), 180 °rotation (R180), 270 °rotation (R270), Y-axis flipped (MY), 90 °Y-axis flipped (MY90), X-axis flipped (MX), 90 °X-axis flipped (MX90). All the rotations are shown in Figure 2.4.

- **NETS:** The following information can be specified to define a net in the netlist for connectivity between components.
  - compName: this metric specifies the components that the net is connecting.
  - pinName: this metric specifies the name of pins for components that the net connects.

The DEF file is strongly connected with the LEF file. Inside the DEF file design and location of components are defined and inside the LEF file physical characteristics of each component are described. For example, as shown in Figure 2.5, a sample LEF and DEF file and associated layout design are given. In this figure, the location of two components (or cells) of OR2x1 and NOR2x1 are given in the DEF file. The location of OR2x1 is at 0 x-axes and 2000 y-axes in the layout, and the location of NOR2x1 is at 1500 x-axes and 0 y-axes in the layout. The size and geometry of each cell are described in the LEF file and highlighted

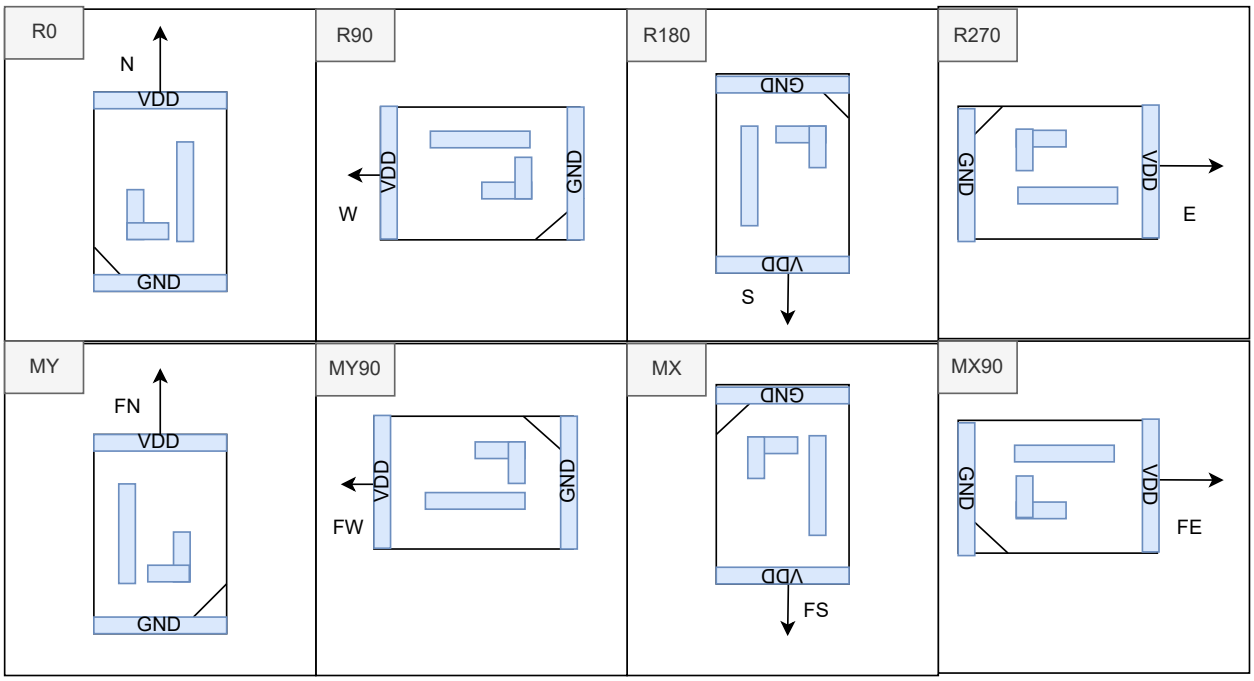


Figure 2.4: Eight possible cell orientations in standard cell design.

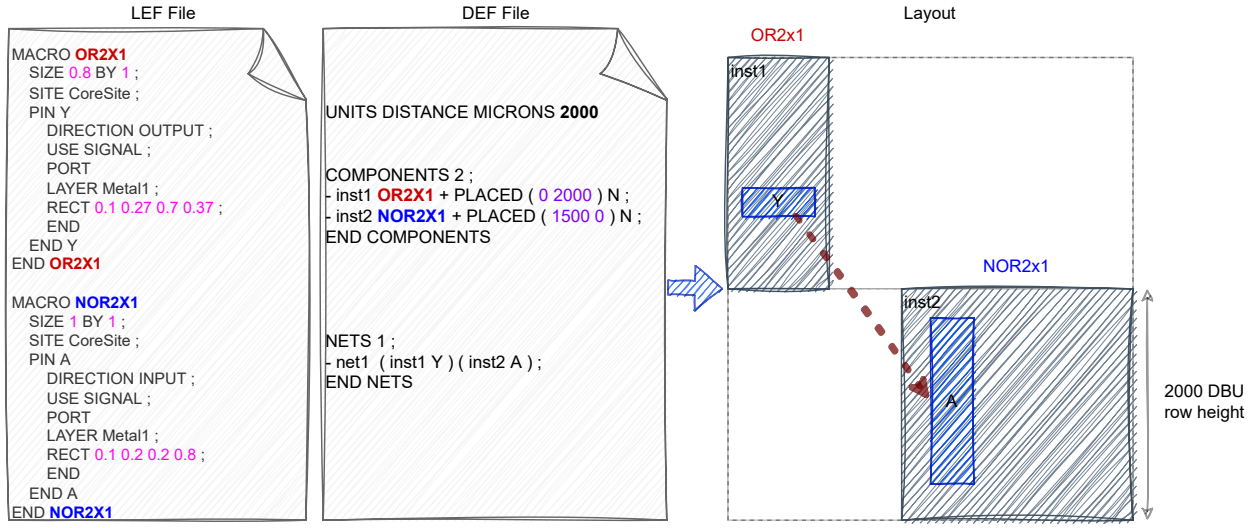


Figure 2.5: An example of LEF and DEF files and associated layout placement.

in pink. By using the information that these two files provided, the layout can be sketched as is shown in Figure 2.5.

The LEF/DEF files that define a place and route design can get extremely large (more than 1 Gigabyte) which represents the number of details and complexity of a design. To parse and handle LEF/DEF files

format the following open sources are available to academics:

- **LEF/DEF parser:** Developed by Synopsys and provides a wrapper to read and write LEF/DEF files [48].
- **Rsyn:** Developed by Rsyn team to provide infrastructure for data modeling and parsing of LEF/DEF files [49].

In this thesis, the infrastructure developed by Rsyn is used to parse and collect information from LEF/DEF files.

2.5 Placement and Routing Background

2.5.1 Placement Background

The main purpose of the placement step is to determine an optimal location for each cell inside the core area such that several objectives like wirelength, reliability, and manufacturability besides placement constraints are satisfied.

The first step of placement is global placement. The algorithms for global placement can be divided into partition-based algorithms (e.g. min-cut placement [50, 51]), analytical algorithms (e.g. quadratic-placement [52, 53]), stochastic algorithms (e.g. simulated annealing [54]), and most recent learning-based methods (e.g. reinforcement learning [18]). The main objective of a global placement engine is to provide a placement solution while optimizing the estimated wirelength, pin density, congestion, and satisfying technology constraints imposed by technology nodes [36, 55–59]. Following that, the legalization goal is to remove possible overlaps between cells and find a legalized location for each cell while minimizing the displacement from the initial global placement solution. A common approach in the legalization step is to use heuristic algorithms by using a local view [60, 61]. In this approach, legalizers move cells to an appropriate location from violation areas by using a greedy search. However, this can impact the performance and the quality of the placement solution due to unnecessarily-large cell movements in high placement density areas. This degradation in the quality of the placement solution is addressed during the detailed placement. The detailed placement incrementally improves the location of each standard cell through local operations such as swapping and shifting.

2.5.1.1 Detailed Placement:

The quality of placement solutions is addressed to improve during the detailed placement by relocating cells in local regions while maintaining legality. To achieve this improvement many detailed placers use

greedy heuristics, graph traversal, and combinatorial optimization algorithms including techniques such as global swap, local reordering, independent set matching, and chain move [37,62–64]. A common objective to optimize during detailed placement is wirelength because less wirelength can help latter steps to cope better with challenges in technology nodes [52].

In the global swapping technique, two cells of the same size swap their location, while they are far from each other in the design. A sample of global swap is displayed in Figure 2.6(a), where c15 and c1 swap their locations. In the local reordering technique, a window is slid in each row and the order of cells is changed to gain wirelength improvement. An example of local reordering is shown in Figure 2.6(b). In this example, the row's order of cells c5 and c6 is swapped. Another technique to explore the solution space and involve more cells is the independent set matching technique [65]. In this technique, the cells that are not connected by a window and the movement of a cell will not impact other cells' wirelength in the set construct a bipartite graph as shown in Figure 2.7(a). In this approach, the wirelength change caused by assigning one cell to the location of another cell is calculated as the weight in the set. The optimal solution can be found by solving a maximum-weighted bipartite matching problem. In this technique also all cells should have the same size. Another approach that involves more cells is the chain move technique [62]. An example of the chain move technique is presented in Figure 2.7(b). In this figure, cell c5 is moved to the location that overlaps with other cells c7 and c8. These two cells are copied to a first-in-first-out queue to find a new candidate position for them such that no overlap exists in the row. These techniques try to find a better solution by perturbing a placement on a small set of cells.

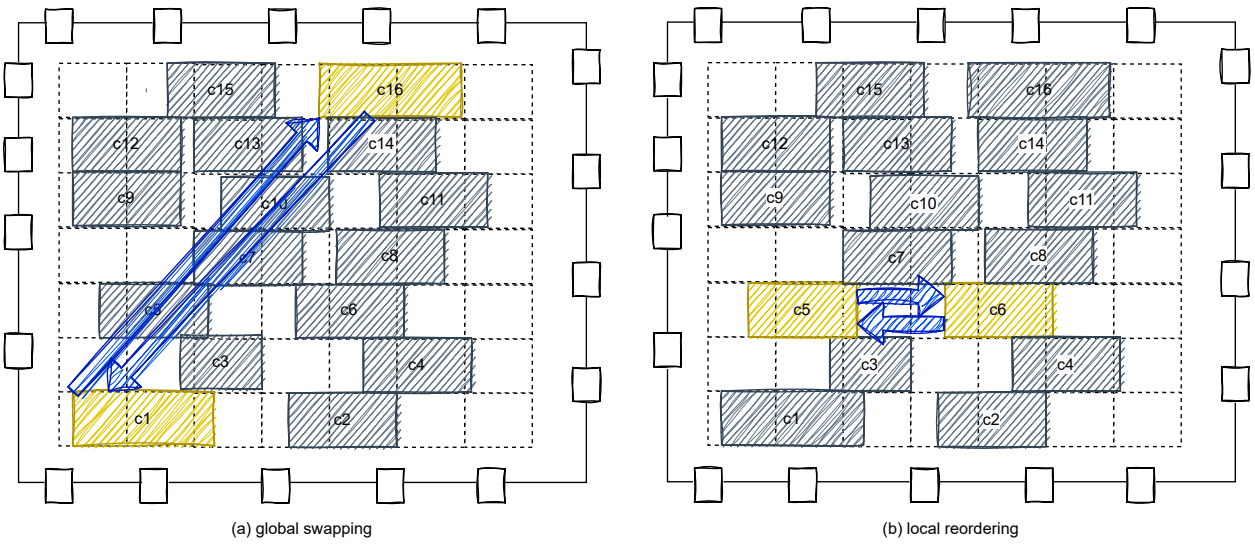


Figure 2.6: The global swapping and local reordering detailed placement techniques.

Besides the improvement of wirelength, these techniques can be used to improve the routability of a

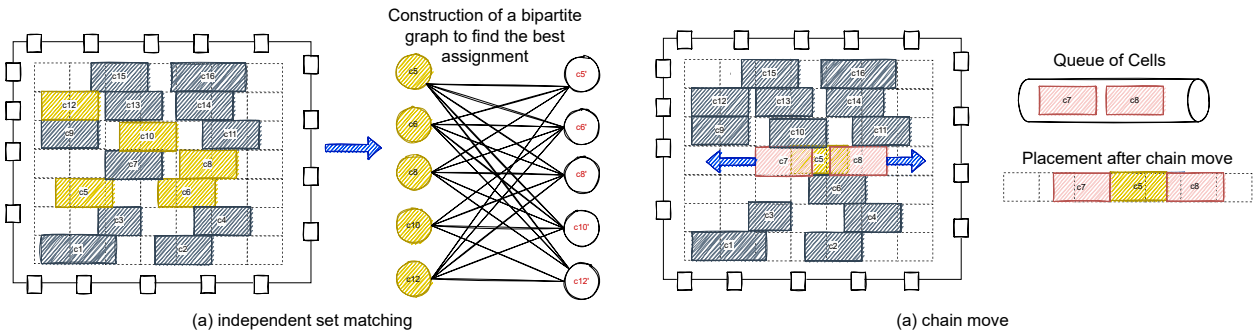


Figure 2.7: The independent set matching and chain move detailed placement techniques.

design by targeting its placement density. Therefore, the detailed placements can be categorized into two groups: Wirelength-Driven Detailed Placement, and Routability-Driven Detailed Placement.

- **Wirelength-Driven Detailed Placement:** Due to the high cost (concerning runtime) of the routing step, simple interconnection models are used to estimate wirelength, for instance, HPWL or Rectilinear 2D Steiner Minimum Tree (RSMT) [36]. Depending on the model used for wirelength estimation the accuracy of the wirelength estimate can be different. For example, for the placement shown in Figure 2.8, the estimated wirelength using HPWL is 12, while if RSMT had been used for the sample placement, the wirelength estimate would have been 15.

  Despite how the wirelength estimation model is defined, there are several techniques that can be used to reduce the defined wirelength estimation model. Sliding a window in a row and optimizing the order of cells inside that window is one technique to improve the wirelength [66]. However, this approach has a narrow scope in wirelength improvement. Fastplace [67] uses global swapping, vertical movement, and local window reordering to move cells toward their optimal region. The optimal region in this work is defined as the median of a cell [68] as shown in Figure 2.9, where an optimal region for cell c1 defines the median of the bounding box of connected cells. To empower the wirelength improvement an expanded window size with branch-and-cut detailed placement is proposed in [69]. In this work, a window within multiple rows that includes 5-8 cells with possible order n! is used, where n is the number of cells. Wing kai et al [61] formulated detailed placement to minimize HPWL by using ICCAD contest 2013 [70] benchmarks. They solve the detailed placement problem in two steps. First, a global movement is used to move cells between G-Cell to optimize displacement constraint. Then, a local reordering was applied to move cells toward their optimal regions. ABCDPlace [37] uses techniques such as independent set matching, global swapping, and local reordering to reduce HPWL in a parallel approach. The above-mentioned techniques are focused to improve HPWL, while they rarely considered 3D routability. A simplified cost model like HPWL can be a poor approximation for

the actual wirelength of the detailed routing solution. This is because it can not estimate multi-layer interconnection issues such as 3D routing congestion and manufacturing constraints.

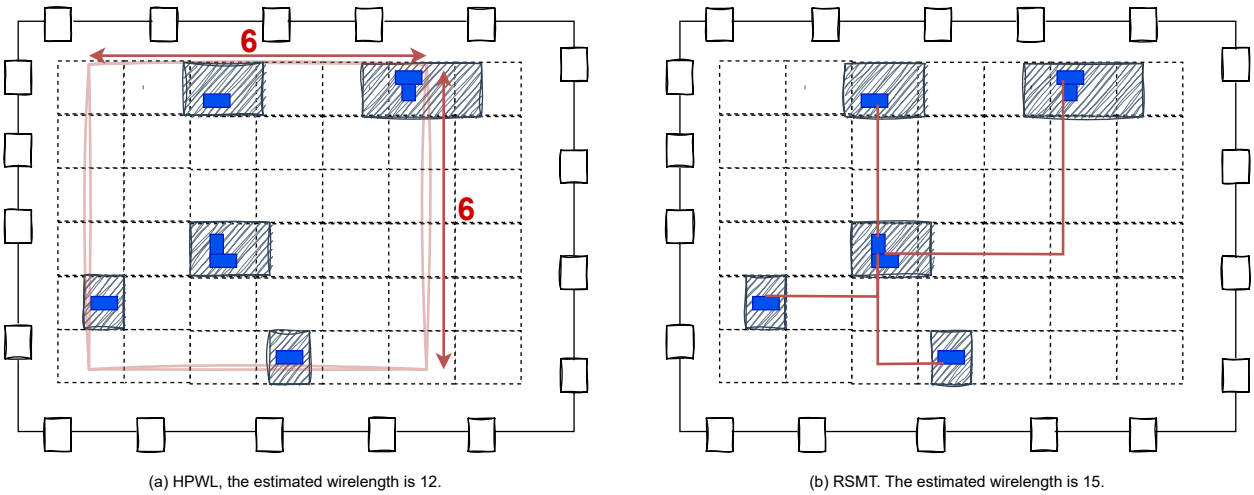


Figure 2.8: Wirelength estimation for a given placement with HPWL and RSMT.

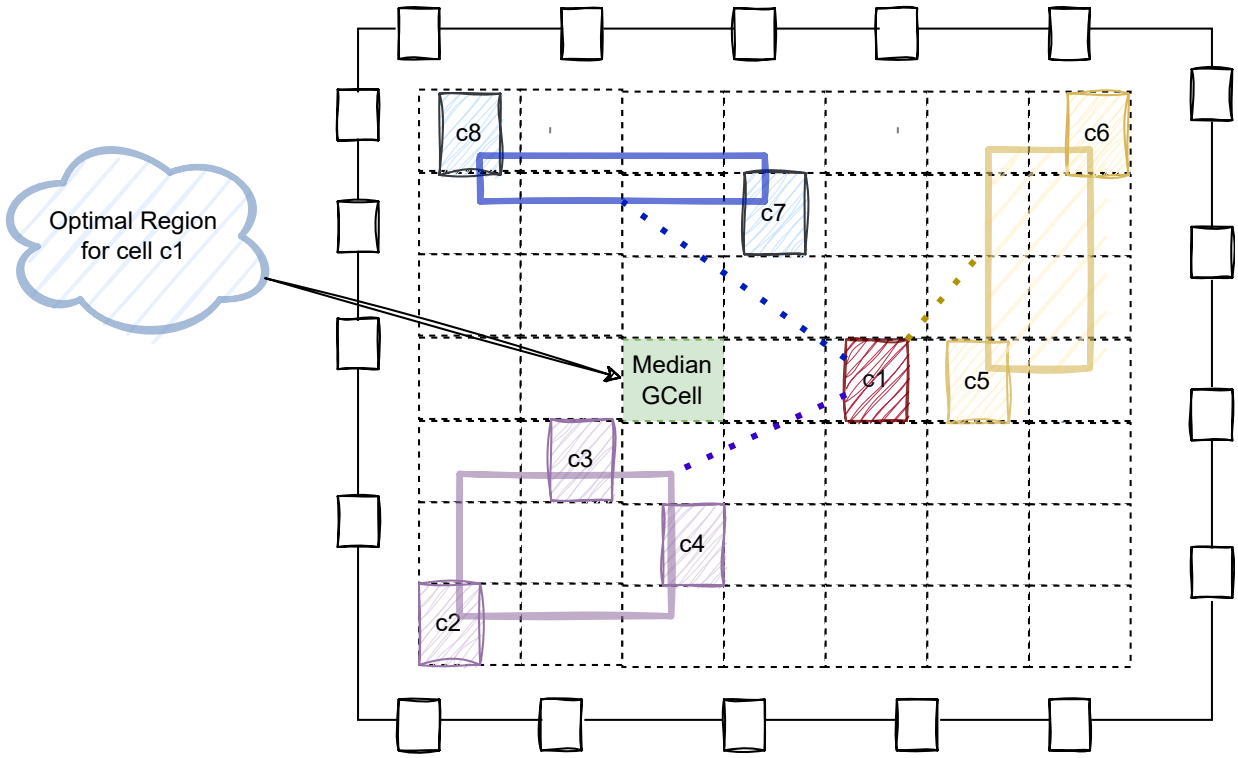


Figure 2.9: An example of an optimal region for cell C1 by using median technique.

- **Routability-Driven Detailed Placement:** To improve routability and 3D interconnection issues (like routing overflow), the global routing solution can be exploited during the detailed placement step [71]. The global router used in the detailed placement step is required to be time-efficient. Therefore,

using heuristic algorithms with predefined patterns like patternRoute [24] can be one approach to using routing feedback during the detailed placement step. There are several known routing patterns used during patternRoute such as L-shape, Z-shape, and U-shape. Besides using the global routing solution during the detailed placement, there are other popular techniques such as white-space injection [72,73] and cell bloating [56,74] to improve the routability. The main goal of these two techniques is to reduce the placement density locally. For example, in the white-space injection technique, some dummy cells (called FILLER) are inserted to reduce placement density and improve routability. Moreover, cell shifting [75] and cell swapping [76,77] are two other common techniques that can be used to mitigate the placement density and improve the routability. Eh?Placer [36] uses a sequential approach to move and swap cells to refine the placement solution. In Eh?Placer, different types of movement including moving to the median, local perturbation, and one-row reordering are used to find the optimal HPWL region for each cell. Also, Eh?Placer promotes routability by performing perturbation during detailed placement by optimizing a cost function that includes wirelength estimation (HPWL), detailed-routing issues, and cell density.

2.5.2 Routing Background

Due to complexity and enormous solution space, the routing step is divided into two sub-steps, named global and detailed routing. Global routing defines the routing regions of each net without going into the details of manufacturing and design rules of the circuit which are handled during the detailed routing stage [1]. In the following, first, the literature work on global and detailed routing is presented and, then, challenges in routing with no cell movement are discussed.

2.5.2.1 Global Routing

A common approach in the global routing problem is to divide the 3D routing area into rectangular grid cells called G-Cells. Then, a coarse-grained 3D grid graph $G(V, E)$ can be defined, where each G-Cell is treated as a vertex ($v \in V$), and an edge is created between every adjacent pair of G-Cells ($e \in E$). Capacities and different constraints are assigned to each edge of G-Cells to optimize the routing topology regarding routability, timing, crosstalk, DRVs, etc. The edge between two adjacent G-Cells in the same layer is called a wire edge. Each wire edge has a capacity defined by the number of tracks available inside the edge. The edge between two G-Cells in adjacent layers with the same 2D coordinates is called a VIA edge.

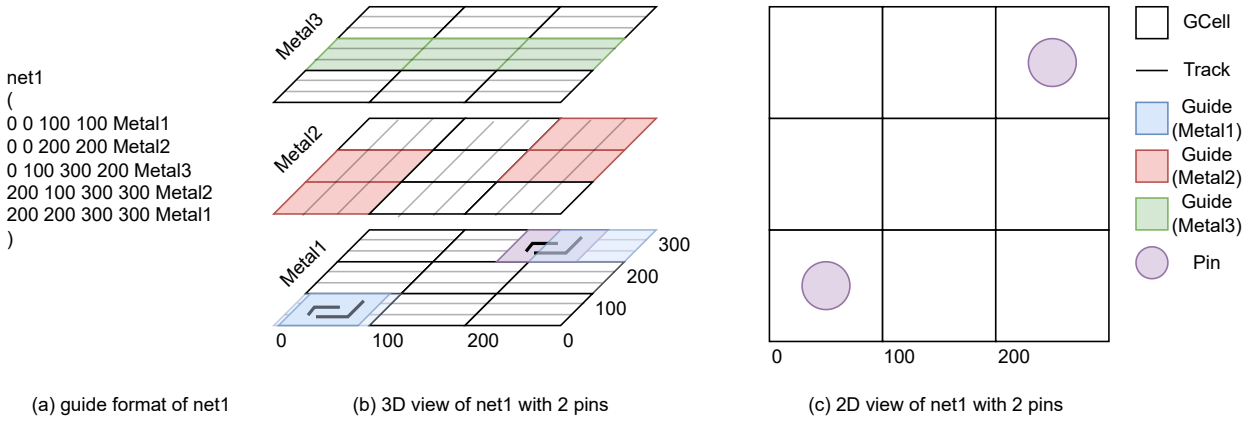
The input of the global routing is a placement solution provided in a DEF file. The output of the global routing is a 3D global routing solution written in a guide file. Then, the detailed router can use this guide

to generate a DRC-clean routing solution. To achieve this, global routing needs to optimize routability and pin accessibility to minimize the disturbance of the routing topology during the detailed routing step.

The guide file format includes a global routing solution where each net consists of several guides. Each guide represents a rectangle covering one or multiple G-Cells in a specific Metal layer. A global routing solution for each net is valid if:

- All pins of the nets are covered by their guides.
- the guides of the net are a connected graph.

A guide file format and a valid 3D and 2D global routing are presented in Figure 2.10(a) and Figure 2.10(b) and (c).



(a) guide format of net1 (b) 3D view of net1 with 2 pins (c) 2D view of net1 with 2 pins

Figure 2.10: Guide input Format with visualization of each guide in 3D and 2D grid graph.

In general, there are two approaches to implementing the global routing step: Applying the route directly to a 3D grid graph and Converting 3D to a 2D grid graph for routing. The first approach is to route nets directly in a 3D grid graph. Although this approach is a strong strategy in routing solution quality, it takes a significant runtime to achieve the results due to the search space. GRIP [78] is an example of this approach, where they employ the Integer Linear Programming model to find the optimum solution for each net. To improve the runtime MGR [79] proposed a multi-level technique to apply a Maze router in a coarse grid graph to reduce the space search.

The second approach in global routing is to convert the 3D grid graph [3] into a 2D grid graph to find a 2D congestion-free routing solution. During the 2D routing step algorithms like Flute [80,81] can be used to generate a Rectilinear Steiner Minimal Tree (RSMT) for each net. Then, in a step called layer assignment, the 2D solution is projected back to the 3D space (shown in Figure 2.11). After that, nets with conflicts

[3] 3D grid graph represents the 3D characteristics of modern Integrated Circuits (ICs)

can be handled by maze algorithms by applying iterations of rip-up and reroute. In this step, to avoid congested routing regions, history costs that model the routing topology history can be employed (shown in Figure 2.12). This approach shows an efficient balance between runtime and solution quality. The global routers that use this approach, I can name CUGR [25], NCTU-GR 2.0 [28], FastRoute 4.0 [24], and NTHU-Route 2.0 [29]. In this thesis, the second approach is used due to the speed and quality of the solution.

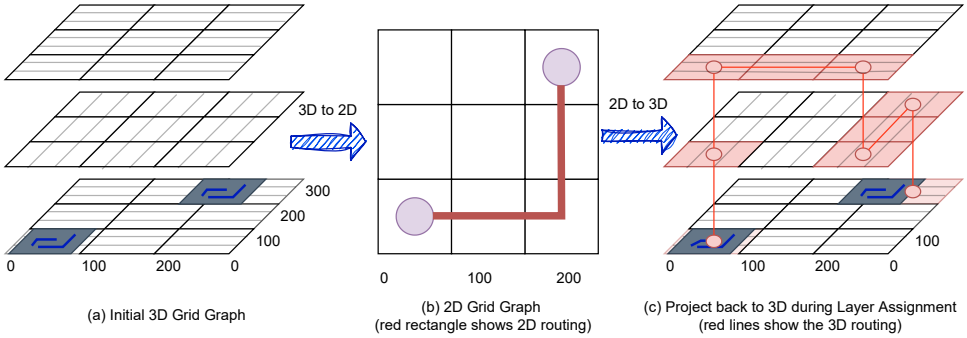


Figure 2.11: Global routing approach to convert 3D to 2D grid graph and write back a 2D routing solution to a 3D global routing solution.

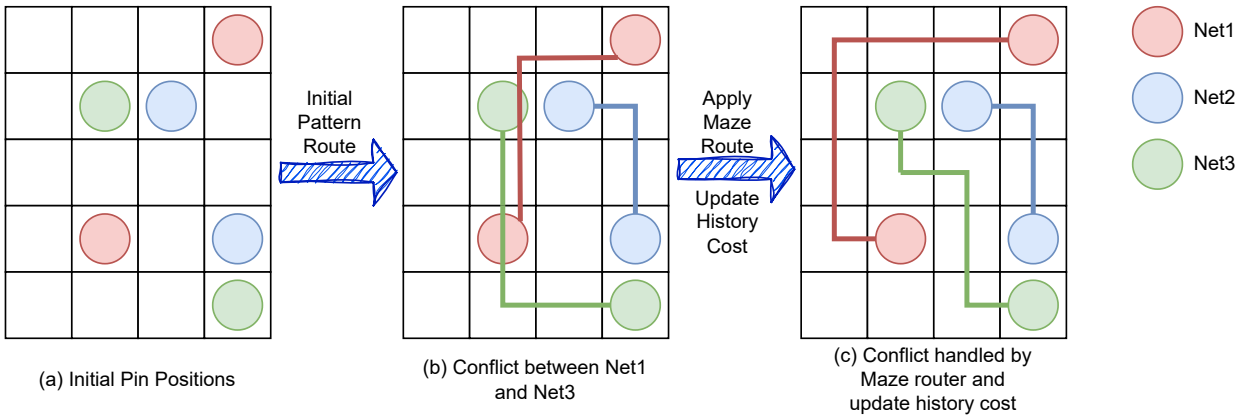


Figure 2.12: Handle conflict nets with Maze algorithm and Update history costs.

2.5.2.2 Detailed Routing

In academics, the detailed routing step can be divided before and after the ISPD 2018 contest. Before the ISPD 2018 contest, only a few works presented end-to-end detailed routing flow. RegualRoute [82] proposes a correct-by-construction methodology, where the routing tracks are formulated as a Maximum Weight Independent Set (MWIS) problem, and it is solved by a heuristic technique. MCFRoute [83, 84] proposes a multi-commodity method, where the detailed routing problem with intricate design rules are formulated

and solved by ILP problem. Rsyn [49] and QRouter [85] are two other detailed routers the provide an initial detailed routing in a sequential approach by using maze algorithms. However, no works, mentioned above, claim a viable solution to the context of real-world IC physical design.

After the ISPD 2018 contest, it was a significant year in the field of detailed routing, where for the first time the ISPD contest released benchmarks that included detailed routing challenges. Although a few works in academics after that could produce some results [86–90], most works could not meet design rule violations. TritonRoute [26, 91] is the only work that could outperform other academic detailed routers and provide an almost violation-free solution. In the TritonRoute engine, which is an open-source platform [92], an extension of the AStar path search in a rip-up and reroute paradigm is used to generate a detailed routing solution. This work also proposes a detailed router worker that can apply to the circuit layout locally to reduce by using parallel processing to achieve a scalable technique. A summary of detailed routers and their methodology with the availability of their source code is given in Table 2.3.

Table 2.3: Available Detailed Routers in Academic

Detailed Router	Year	Methodology	Open-source
RegularRoute	2011	Trunk Router, AStar, MWIS	No
MCFRoute	2016	Multi-commodity	No
QRouter	2017	Maze Algorithm	Yes
Rsyn	2017	AStar Algorithm	Yes
DRAPS	2019	AStar-interval-based fast path search algorithm	No
Multithread-DR	2019	AStar with negotiation-based rip-up and reroute schema	No
Dr. Cu	2020	Multi-level data structure with Maze algorithm	Yes
TritonRoute	2020	Extended AStar Algorithm with rip-up and reroute	Yes

2.5.2.3 Routing Technology Constraints

The routing constraints in the physical design optimization can be categorized as follows:

- **Electrical Constraints:** These constraints ensure the electrical behavior of the design such as signal connectivity, delays, and timing constraints such as open and short signals. The electrical constraints have the highest priority to be obeyed. These constraints guarantee the validity of the signals and wires that can be implemented. There are two connectivity Constraints Open and Short addressed in this thesis.
  - **Open:** An open violation happens if all the pins in a net defined in DEF files are not fully connected.
  - **Short:** A short violation happens if two independent net crosses each other in the same layer.

- **Technology Constraints:** The technology constraints define the manufacturing requirements, and they depend on the technology node. In this thesis, two routing rules are investigated: Spacing and min-area.
  - **Spacing**: Spacing rules define that wires can not be too close to each other due to parasite effects they can damage each other's logic.
  - **MinArea:** The minimum size of each component in the routing can not be smaller than minArea. An example of minArea violation and a solution to solve the violation is shown in Figure 2.13. In this figure, the minArea is 0.01 micrometer (um) while the wire area is 0.008. Since the area of the wire is less than the layer's minArea, a minArea violation happens in Figure 2.13 (a). To solve minArea, one solution is to add a metal extension called Patch to one edge of the wire as is shown in Figure 2.13 (b).

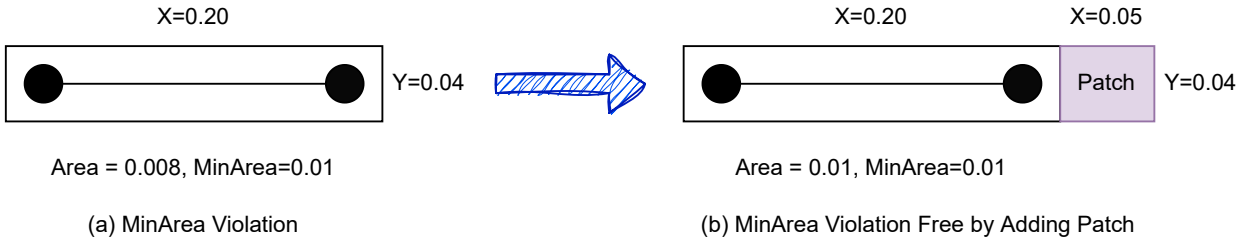


Figure 2.13: Handle minArea violation by adding a metal extension called Patch: (a) minArea violation, (b) minArea violation free.

- **Geometry Constraints:** These are introduced to reduce the complexity of the design process. They include routing preference metrics such as a preferred wiring direction during routing or the placement of cells in the rows. There are several metrics to evaluate the quality of the detailed routing solution. These metrics are not hard rules but strong preferences for satisfying their results in a better routing solution in terms of timing, routability, and manufacturability. It is common that the global routing results are optimized for certain metrics like congestion, therefore, a detailed router requires honoring the global routing solution as much as possible to avoid disturbance in the solution. The following metrics are used to show detailed routing respecting the associated routing regions and the routing grid by the global routing solution and the design's description, respectively.
  - **Out-of-Guide Wiring (OFGW):** If the center of a wire is not inside the associated routing region, it is considered an Out-of-Guide Wiring (OFGW) (shown in Figure 2.14).
  - **Out-of-Guide VIA (OFGV):** If the center of a VIA is not inside the associated routing region, it is considered an Out-of-Guide VIA (OFGV) (shown in Figure 2.14).

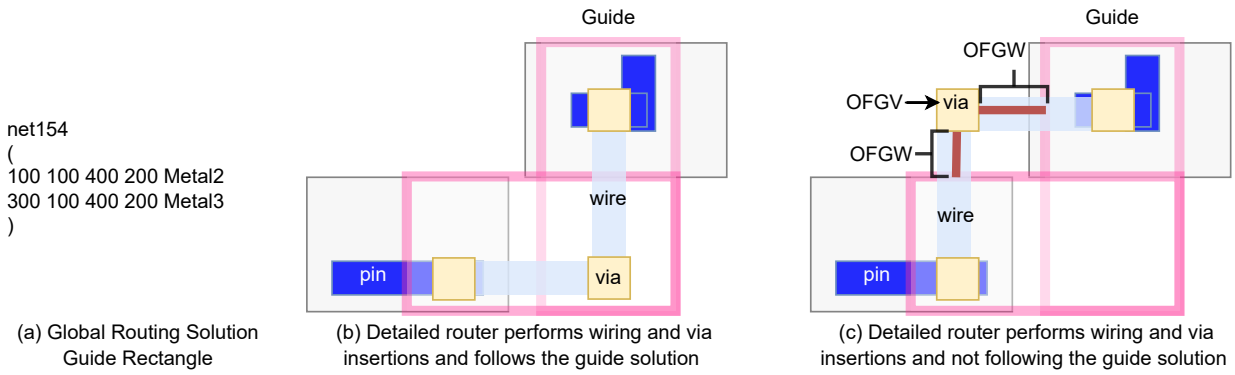


Figure 2.14: An example of OFGW and OFGV: (a) shows a guide solution provided by global routing. (b) shows detailed routing following the guide solution. (c) shows detailed routing doesn't follow the guide solution and causes OFGW and OFGV.

– **Out-of-Track Wire (OFTW):** Each metal has a predefined routing path called Track defined in the DEF file. The routing wires aligned to the tracks are called on-track otherwise it is off-track (shown in Figure 2.15).

– **Out-of-Track VIA (OFTV):** If the center of a VIA has an intersection with tracks is called on-track VIA otherwise it is off-track VIA (shown in Figure 2.15).

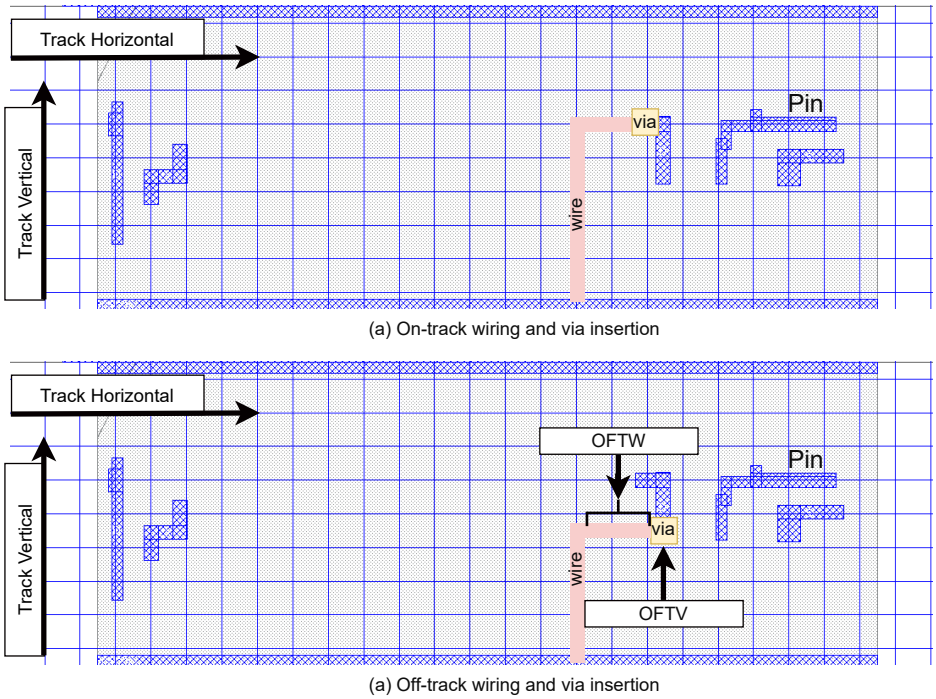


Figure 2.15: On-Track and Off-Track routing constraint wiring and via insertion are shown in figures (a) and (b).

– **Wrong Way Wiring (WWW):** Each metal layer has a preferred routing direction. If a wire does not respect the routing direction of the layer, it is called wrong-way wiring (shown in Figure 2.16).

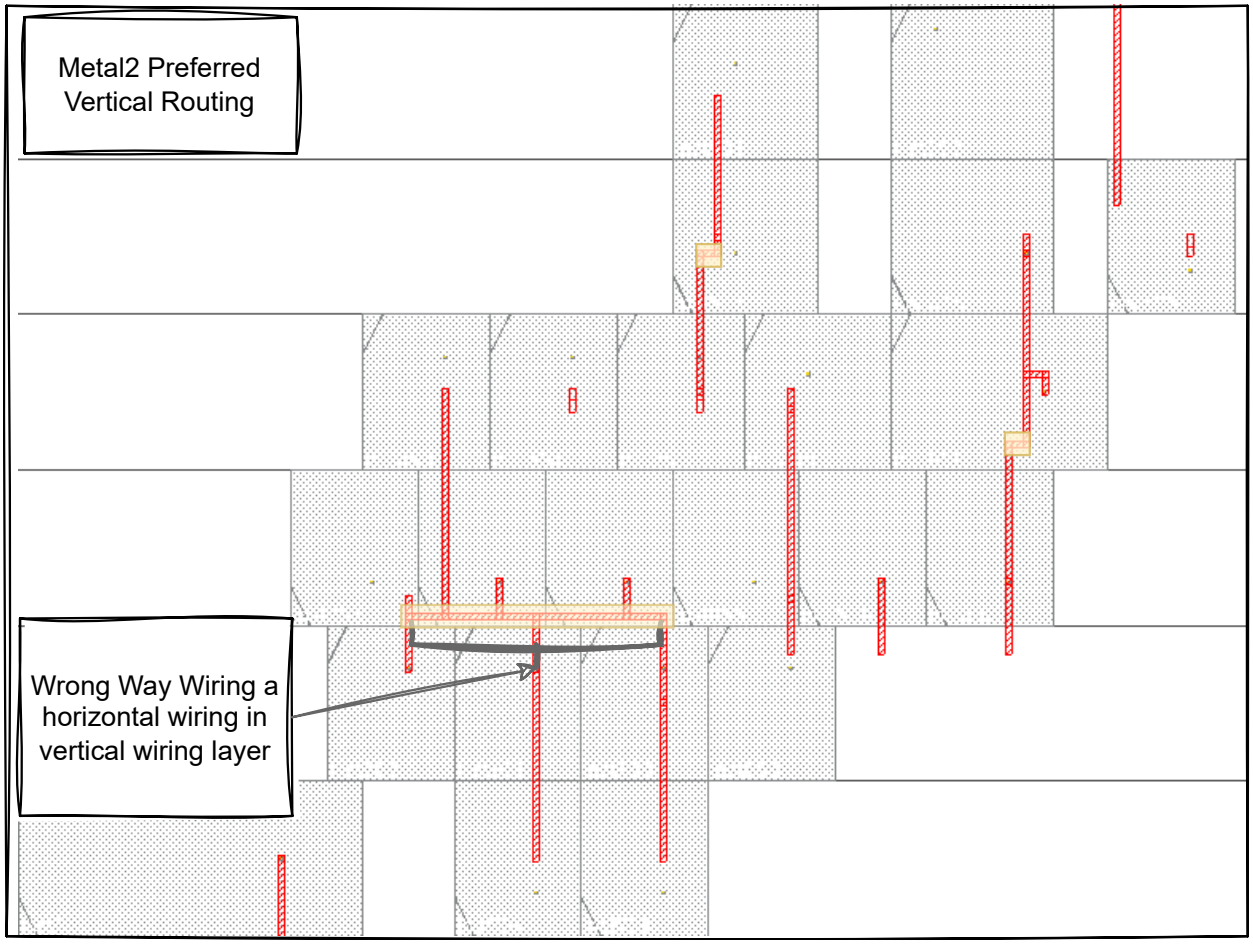


Figure 2.16: An example of Wrong Way Wiring in Metal2 with preferred vertical routing direction.

2.6 Cooperation Routing and Placement Background

Due to the high number of issues arriving from the ever-increasing interconnection complexity in advanced technology nodes, adding cooperation between the routing and placement steps has become a popular approach to improve the routing solution quality. This technique is the main motivation of the ICCAD Contests 2020 and 2021 (Problem. B), where both target the development of "routing with cell movement" techniques [34, 35]. These contests have pointed out the need for putting more effort towards integrating the routing and placement engines. The cooperation between routing and placement can be done in two approaches:

- **Using routing topology (routing edges) to improve the placement:** In this approach, an initial

routing is performed by a routing algorithm. Then, optimal movement locations are found by using the routing topology. However, a high dependency between nets is one of the challenges in this approach, where improving one net can degrade the topology routing of other nets. Also, another challenge of this approach is the development complexity.

- **Performing routing for different placements and selecting the best routing solution:** In this approach, multiple placements with their associated routing can be generated that optimize different objectives. Then, the best routing solution can be selected. This approach is more straightforward than the first one. However, performing routing for each placement is a runtime expensive task.

In recent literature, Starfish [93] proposed a partial rerouting technique to combine routing with cell movement. They developed a multi-source single target AStar algorithm, that connects moved cells back to the trunk of a previous routing topology. A dynamic threshold is proposed by this work to improve the gain from each movement. Moreover, to pick up a good candidate destination, a lookup table is proposed for the gain estimation of cell movement. Although this look-up table uses min-routing-layer and preferred routing direction constraints to estimate movement gain, it is not clear how accurately this look-up table can perform in benchmarks with advanced technology nodes and detailed routing information. Huang et al [94] proposed a Breadth-First Search (BFS)-based approximation with routing constraints such as layer direction, minimum layer, VIA reused, and overflow to reduce the optimal region for cell movement. This work also used AStar to do partial rerouting after cell movement. Peng et al [95] used rough global routing results produced by SRP [96] to estimate optimal locations for each cell's movement. Then, an incremental strategy is proposed to move cells and reroute associated nets. Ophidian [97] proposed an incremental movement routing. First, all the cells are sorted in descending order according to the total wirelength of the nets connected to them to address the large nets first and further reduce wirelength with fewer movements. In order to find the optimum position for cell movement, nine different movement candidates for each cell are chosen: the median point of the connected nets; the four neighbors of the median point; the four neighbors of the initial location of the cell. In Ophidian, cells are moved to their optimal regions and then routed by AStar. If the total routing wirelength after each movement improved, the movement would be kept otherwise cell will return to its initial position.

All above mentioned works Ophidian [97], Starfish [93], Huang et al [94], and Peng et al [95] used ICCAD benchmarks to evaluate their techniques. During the ICCAD 2020 contest, ten benchmarks were released, where each test circuit was described using a text file. Each circuit's description includes a list of cells, nets, routing layers, and G-Cell grids boundary with an initial placement and routing. The statistics of the ICCAD 2020 benchmarks are illustrated in Table 2.4. In this Table, Columns 1 to 6 represent the name of

Table 2.4: ICCAD 2020 circuit statistics. Columns 1 to 6 represents the name of each circuit, the number of nets, the number of cells, the routing layers, the size of grid, and the initial routing wirelength (WL). Note: The WL in ICCAD 2020 is calculated according to the number of GCells.

Benchmarks	Nets (#)	Cells (#)	L (#)	Grid Size	Initial Est. WL (GCell)
case1	6	8	3	5x5	64
case2	6	6	3	4x4	30
case3	2,644	2,735	7	27x33	32,600
case4	179,996	204,206	12	277x277	4,680,681
case5	92,546	96,682	16	104x103	1,763,627
case6	332,080	352,269	16	237x236	7,188,481
case3B	2,563	2,604	7	29x29	29,748
case4B	183,137	207,347	12	277x277	4,886,698
case5B	92,559	96,689	16	104x103	1,721,530
case6B	332,045	352,234	16	237x236	7,340,802

Table 2.5: Comparing the results of wirelength improvement on ICCAD 2020 benchmarks. Four flows are compared such as ophidian [97], Starfish [93], Huang et al [94], and Peng et al [95].

Benchmark	Initial	Wirelength				% Improvement of Wirelength			
	Wirelength	Ophidian	Starfish	Huang et al	Peng et al	Ophidian	Starfish	Huang et al	Peng et al
case03	33k	23k	21k	21k	21k	29.20	**35.61**	35.50	35.45
case04	4681k	2849k	2616k	2617k	2615k	39.13	44.11	44.10	**44.13**
case05	1764k	1251k	1064k	1068k	1067k	29.05	**39.67**	39.43	39.47
case06	7188k	4925k	4451k	4451k	4437k	31.49	38.08	38.08	**38.27**
case03B	30k	20k	18k	18k	19k	31.70	**38.08**	38.02	37.65
case04B	4887k	2958k	2686k	2690k	2686k	39.47	**45.04**	44.95	**45.04**
case05B	1722k	1239k	1053k	1057k	1055k	28.02	**38.82**	38.60	38.70
case06B	7341k	5316k	4556k	4554k	4535k	27.58	37.94	37.97	**38.22**

each circuit, the number of nets, the number of cells, the routing layers, the size of the grid, and the initial routing wirelength (WL). As can be seen, the number of nets varies from 6 to 332k nets from the smallest to the biggest circuit in this set of benchmarks. The results of each work are shown in Table 2.5. All techniques showed a significant improvement in wirelength compared to the initial wirelength provided by the contest. For example, in case04B Starfish could improve the routing by 45% or the work proposed by Peng improved wirelength by 38% in case6B.

The ICCAD 2020 contest was the main contest with the topic of routing with cell movement. Due to the complexity of combining the routing engine with cell movement, the benchmarks are simplified and far from realistic advanced technology node challenges.

Since there is no detailed routing information in these ICCAD benchmark sets, therefore, there is no physical wiring and VIA insertion, nor complex detailed routing rules. This means that the score in ICCAD is the wirelength estimation, and it is not an actual wirelength. Tiago et al. [98] used a cluster-based technique, where for each cell its median with related route is calculated. Then by using an ILP model, they find the best placement and routing arrangement. They evaluated their techniques on ISPD 2018 contest benchmarks [42]. Although in this work, a significant improvement in estimated VIAs and wirelength in a

global routing solution is reported, it is not clear what is the consequence of that in the detailed routing solution. All the above-mentioned literature tried to improve the estimated wirelgnth without considering modern technology nodes challenges, therefore, several important questions will come up:

- Can I improve an already optimized detailed routing solution in the benchmarks with detailed routing information by using routing with cell movement?
- What challenges will the development of routing with cell movement face in benchmarks with advanced technology nodes?
- What would be the effect of only improving estimated wirelength on other metrics such as VIA insertion and design rule violations?

To answer the above questions, the necessity of evaluating the techniques in benchmarks that simulate the modern technology node challenges is essential.

2.7 Challenges in Cooperation between Routing and Placement

The development of a framework that integrates routing with cell movements has several challenges. The first challenge is the complexity of development. This includes the infrastructure development for combining the optimization engines that can impose new constraints on the problem. For example, in the state-of-the-art routers, the pin access locations are fixed and it is calculated once after the placement engine. However, when routing and placement both are changing at the same time the location of pins is required to be updated after each movement. The second challenge is to show the efficiency of the proposed framework that can improve the detailed routing solution in advanced technology nodes. This requires benchmarks that have detailed routing information such as advanced design rules and physical track characteristics. These details were removed from the ICCAD 2020 [34] and ICCAD 2021 [35] benchmarks, therefore, using the proposed benchmarks suites will not guarantee the efficiency of the proposed flow. In addition to that, the feasibility of the proposed framework is crucial, and it is the main gap in the state-of-the-art. The runtime plays a dominant role in the feasibility of the proposed approach. Evaluating the quality of each movement by routing is a runtime expensive task. Moreover, due to complexity and dependency, this is a hard problem to parallelize. Therefore, a framework that can cooperate with routing and placement and shows improvement in the detailed routing solution in modern benchmarks with advanced technology constraints is essential.

2.8 Summary

In this chapter, the terminology and technical concepts used to describe the place and route tools in EDA are explained first. Then, the input and output of the problem with the main constraints for the cooperation between routing and placement are presented. After that, to have a better understanding of the placement and routing steps of the physical design a review of popular methods in each step is presented. Finally, the background, open gaps, and challenges to the development of cooperation between routing and placement are presented.

Chapter 3

Background: Basic Algorithms and Optimization

3.1 Introduction

The concepts presented in this chapter are the preliminaries of methods developed in this thesis to address placement and routing challenges mentioned in chapter 2. There are three main techniques used in this thesis to solve the physical design problems: dynamic programming technique, ILP optimization technique, and heuristic techniques.

This chapter is organized as follows: First, the main concepts for complexity analysis are provided in Section 3.2. The preliminary concepts of graph theory that are used in this thesis are defined in Section 3.3. This is followed by Section 3.4, the main data structure used to organize and handle the query process of the layout. In Section 3.5, the sequence of steps and main properties of the dynamic programming technique are discussed. Then, the ILP model to solve optimization problems is explained in Section 3.6. In Section 3.7, the heuristic methods are illustrated. Finally, a summary of this chapter is provided in Section 3.8.

3.2 Complexity Analysis Concepts

A sequence of steps that are taken to solve a complex problem is called an algorithm. An asymptotic notation to evaluate the performance of an algorithm can be expressed by big-O notation. This notation expresses how the runtime scales as the size of the input increases. For example, a linear time algorithm with the size of N can be notated by O(N). An example of a comparison between different algorithms' performance from

the best to worst is represented in Figure 3.1. In this figure, the $O(logN)$ has the best performance, and the $O(N!)$ has the worst one. In other words, this figure shows how each algorithm's runtime scales with increasing the size of input data.

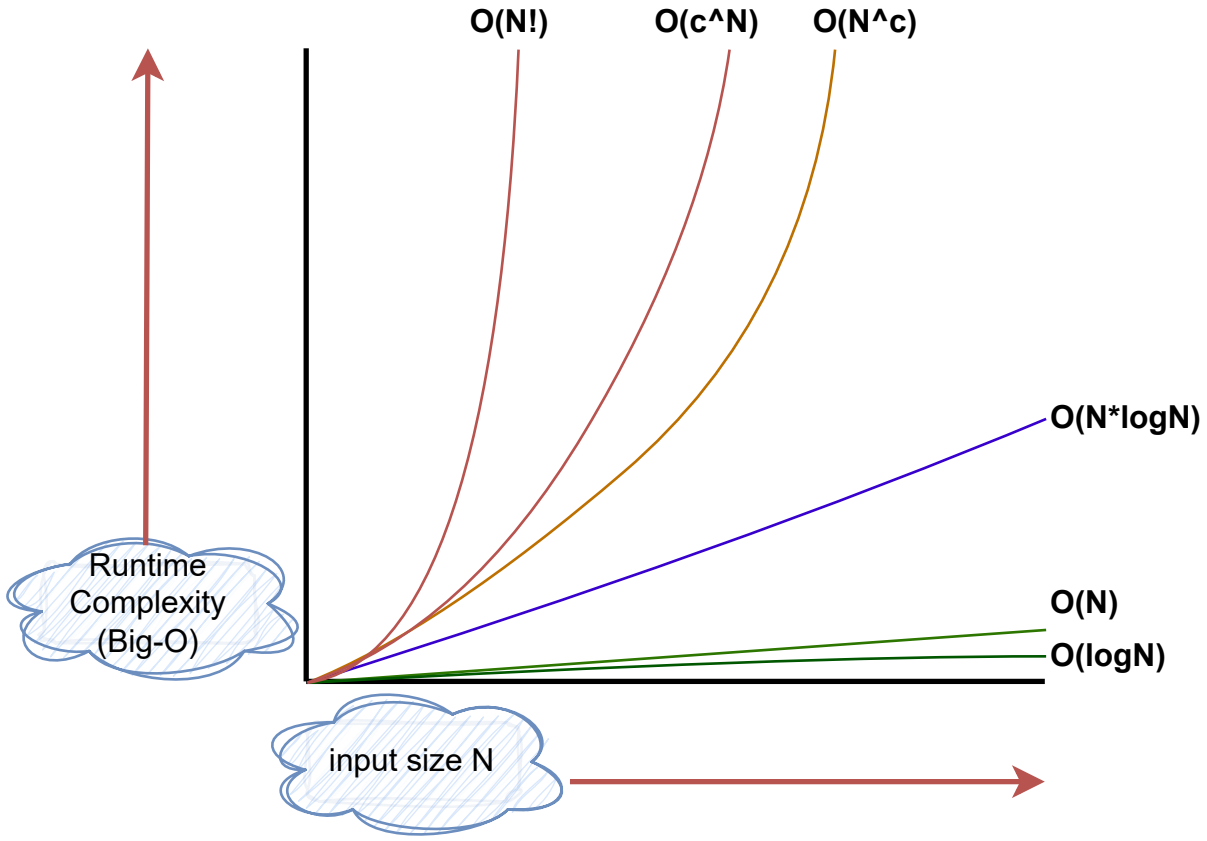


Figure 3.1: An example of the Runtime scaling by using Big-O notation.

3.3 Graph Theory

Graph theory was used to solve the Konigsberg bridge problem in 1736 [99] for the first time. A graph can be described by $G = (V, E)$, where V is the set of vertices and E is the set of edges between the vertices. Graph theory is widely used in physical design to abstract various problems such as routing and placement. The following is a list of basic terms used in this thesis from graph theory.

Definition 3.1 (Undirected graph)**.** The undirected graph is a graph that is built of a set of vertices that are connected by bidirectional edges. An undirected graph is shown in Figure 3.2(b).

Definition 3.2 (Directed graph)**.** The directed graph is a graph that is built of a set of vertices that are connected by directed edges. If there is at least one directed cycle in a directed graph, it is called a directed

cyclic graph. Otherwise, it is called a directed acyclic graph. A directed graph is shown in Figure 3.2(c).

Definition 3.3 (Connected Graph)**.** A graph with at least one path between each pair of vertices.

Definition 3.4 (Path)**.** A path is a sequence of edges that makes a connection between two nodes such that no vertex is repeated. A path from nodes a to f is shown in Figure 3.2(a).

Definition 3.5 (Tree)**.** A tree is a graph in which n vertices are connected by n-1 edges. The trees can be undirected or directed. Two terms are used for describing vertices in a tree: root or leaf. A root vertex (or node) is a node with no incoming edge. Any node with one edge is called a leaf, while it has no outgoing edge. A tree is shown in Figure 3.2(d).

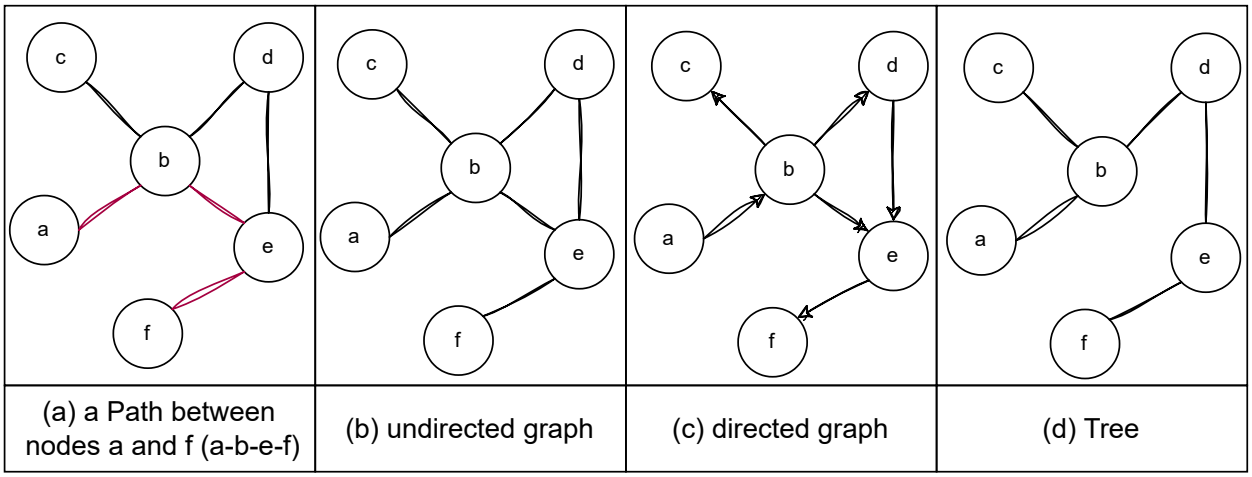


Figure 3.2: Graph theory terminology.

Definition 3.6 (Spanning Tree)**.** A spanning tree in graph G is a connected and acyclic subgraph that includes all vertices of graph G.

Definition 3.7 (Minimum Spanning Tree)**.** A Minimum Spanning Tree (MST) is a spanning tree with the smallest sum of edge length.

Definition 3.8 (Rectilinear Steiner Minimum Tree)**.** A Rectilinear Steiner Minimum Tree (RSMT) is a tree with total minimum edge length that is calculated in Manhattan (i.e. rectilinear) distance. The RSMT uses some extra points (i.e. Steiner points) to reduce the total edge cost. A sample of 2D routing with RSMT is shown in Figure 3.3.

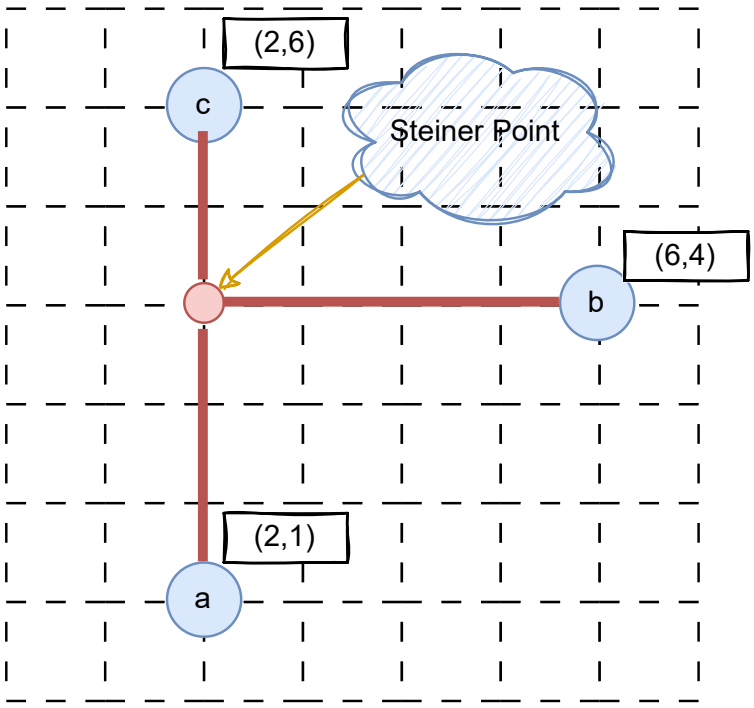


Figure 3.3: A sample of 2D routing with Rectilinear Steiner Minimum Tree.

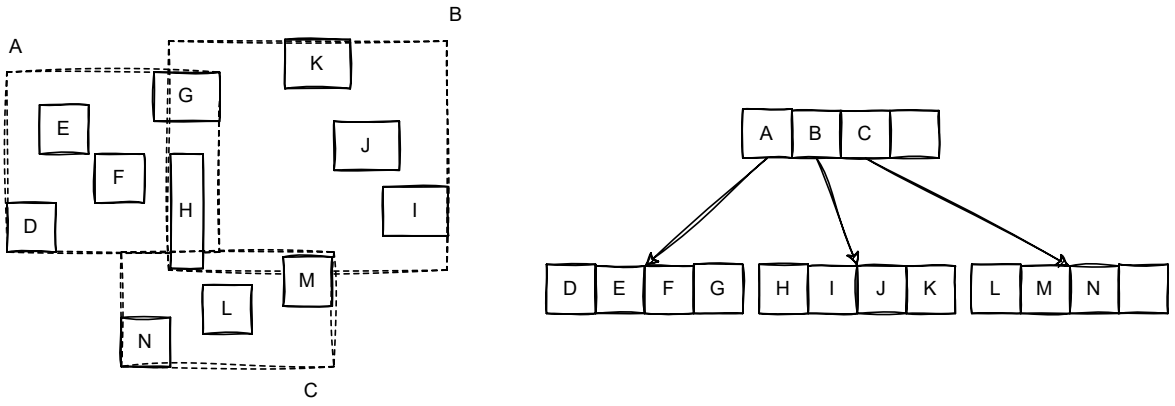
3.4 Data Structure

There is a wide range of data structures used in the development of this thesis such as arrays, binary trees, hash tables, and graphs. However, because most of the design components such as cells are represented as rectangles, an R-Tree data structure is used to support main operations such as finding the intersected rectangles (or components) in the layout. Thanks to that fast O(log(n)) area queries are possible [100].

3.4.1 R-Tree

The R-tree data structure was designed by Guttman [101] to increase the efficiency of organizing the large set of 2D rectangles in VLSI applications. The R-tree organizes the 2D rectangles by representing them by their Minimum Bounding Rectangles (MBRs). Each node of the R-tree corresponds to an MBR. An example of a set of 2D rectangles organized by an R-Tree is shown in Figure 3.4. In this figure, a set of MBRs (D, E, F, G, H, I, J, K, L, M, and N) are organized at the leaf level of the R-tree. In this figure, the MBRs (A, B, and C) also stored the aforementioned MBRs as their child nodes. As can be seen, in Figure 3.4(b), R-tree is a hierarchical data structure and it is formed on B+-tree [100].

To improve the performance and efficiency of the query process in the R-tree, there are several heuristic techniques used in basic operations such as insertion, deletion, and search to minimize the overlapping MBRs and reduce their size. These techniques make R-tree a well-suited data structure to store components of



(a) A sample of rectangles with their minimum bounding rectangles

(b) Corresponding R-tree

Figure 3.4: An example of a list of rectangles stored in an R-tree.

physical design, and it can help designers to have an efficient query process during the implementation of physical design algorithms. In this thesis, the R-tree data structure is implemented by using the Boost library [102].

3.5 Dynamic Programming Technique (DPT)

Dynamic programming, introduced by Richard Bellman in 1953 [99], is a technique for finding an optimal solution. This technique solves problems by breaking them down into sub-problems, and it can benefit from memory to reduce the runtime of algorithms. There are three main steps required to be taken to solve a problem by using dynamic programming. The sequence of these steps are:

1. A problem is broken into smaller sub-problems.
2. The sub-problems are solved optimally.
3. The optimal solutions of sub-problems are combined to find a solution for the original problem.

To find an optimal solution by dynamic programming, the problem must have two properties: Optimal substructure and overlapping sub-problems. The term optimal substructure defines that the optimal solution of the sub-problems can be utilized to achieve the optimal solution of the original problem. The term overlapping sub-problem defines that to solve a problem with size N, we must solve a sub-problem of size N-1.

Several physical design problems such as the layer assignment problem during global routing can be solved optimally by using dynamic programming [24, 25, 103–105]. The complexity of dynamic programming can

be calculated by $O(n * S)$, where n is the number of states that dynamic programming considers multiplied by the time taken per state, S.

3.6 ILP Problems

An optimization problem can be described as follows [106]:

$$\text{Minimize (or maximize) } f_o(x) \tag{3.1}$$

such that

$$f_i(x) \leq b_i,\ i = 1, ..., n \tag{3.2}$$

The vector $x = (x_1, ..., x_n)$ contains decision variables of the optimization problem. The $f_o : R^n \rightarrow R$ is an objective function that takes in n real numbers as input and produces a single real number as output. The functions $f_i : R^n \rightarrow R$ for $i = 1, \cdots, m$ are constraint functions. The constants b_i define the limits and boundaries of each constraint function. An optimal solution is notated by $x*$ if for any vector of z with $f_1(z) \leq b_1, \cdots, f_m(z) \leq b_m$, the objective function of $f_o(x^*) \leq f_o(z)$ when the objective is minimization and $f_o(x^*) \geq f_o(z)$ when the objective is maximization.

The characteristics of the objective, constraints, and domain of an optimization problem define the class of optimization. If both objective and constraints are linear and their decision variables are integers. Then, the optimization problem is classified as an ILP optimization problem. An ILP problem can be written in the following general form:

$$\text{Minimize (or maximize) } c_1 \times x_1 + c_2 \times x_2 + \cdots + c_n \times x_n \tag{3.3}$$

s.t.

$$\begin{gathered} a_{1,1} \times x_1 + a_{1,2} \times x_2 + \cdots + a_{1,n} \times x_n \leq b_1 \\ a_{2,1} \times x_1 + a_{2,2} \times x_2 + \cdots + a_{2,n} \times x_n \leq b_2 \\ \vdots \\ a_{m,1} \times x_1 + a_{m,2} \times x_2 + \cdots + a_{m,n} \times x_n \leq b_m \end{gathered} \tag{3.4}$$

The decision variables $x_1, x_2, \ldots, x_n \in \mathbb{Z}$ are required to be integers. $c_1, c_2, \cdots, c_n$ are coefficients that multiply the decision variables in the objective function. The constraints are represented by the inequalities on the right-hand side, where $a_{i,j}$ are coefficients. $b_1, b_2, \cdots, b_m$ are the boundaries of each constraint function.

For example, consider the following ILP problem:

$$\text{Minimize } 3 \times x_1 + 4 \times x_2 \tag{3.5}$$

s.t.

$$\begin{gathered} 2 \times x_1 + 3 \times x_2 \leq 4 \\ -1 \times x_1 + 2 \times x_2 \leq 3 \\ x1, x2 \geq 0 \\ \mathrm{x}_1,\, x_2 \in \mathbb{Z} \end{gathered} \tag{3.6}$$

In this example, the decision variables are x_1 and x_2, which are required to be integers. The objective function is to minimize $3 \times x_1 + 4 \times x_2$. The constraints are $2 \times x_1 + 3 \times x_2 \leq 4$, $-1 \times x_1 + 2 \times x_2 \leq 3$, and $x1, x2 \geq 0$, which specify the range of possible values for the decision variables. To solve this ILP problem, we would need to find the values of x_1 and x_2 that minimize the objective function while satisfying the constraints.

There are two main methods for solving ILP problems: those based on Dantzig's Simplex [107,108], and more recently, interior point methods [109]. The complexity of the interior point is n^2m, where n is the number of variables and m is the number of constraints in the ILP problem.

These techniques although can solve a problem with 100 variables and 1000 constraints in a relatively short amount of time by a typical desktop Personal Computer (PC) in 2023, their runtime with extremely large problems is still a challenge and makes them impractical. Note: The details of interior-point and simplex algorithms are out of the scope of this thesis. There are well-known tools that have implemented

these algorithms efficiently such as CPLEX [110] and Gurobi [111]. In this thesis, CPLEX tool is used to solve ILP problems.

3.7 Heuristic Methods

The placement and routing problems are NP-hard (Section 3.2) [1]. Heuristic algorithms are a common approach in physical design problems to achieve an approximate solution in a practical time. The quality of each heuristic algorithm is evaluated based on runtime and the final quality of the solution.

Heuristic algorithms can be classified into two categories: deterministic and stochastic. In the deterministic techniques, all the decisions made by an algorithm are repeatable (like Dijkstra [112] and AStar [113]). Stochastic algorithms on the other hand use randomly chosen decisions. In other words, two runs of the algorithm may provide different solutions (like Simulated annealing [54]).

In addition to that, the structure of heuristic algorithms can be either constructive or iterative. In a constructive approach, the heuristic algorithm starts with an initial solution and adds components to complete the solution. While, in an iterative approach, the heuristic starts with a complete solution and tries to improve the current solution over iterations until one or more stop criteria have been satisfied. As shown in the Figure 3.5, in the physical design algorithms is common to employ both algorithms to achieve a complete solution by initially employing a constructive approach and then refining the solution used by an iterative algorithm.

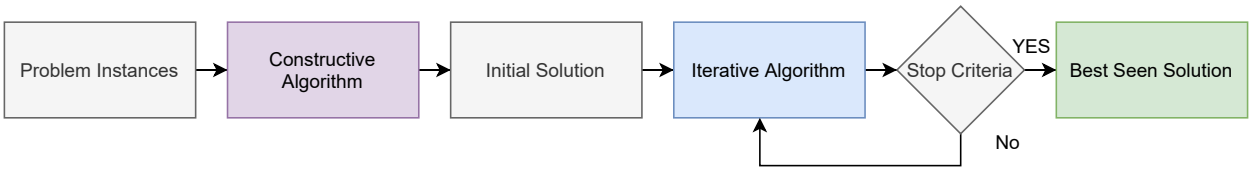


Figure 3.5: It is a common approach in physical design to use a constructive heuristic algorithm to find an initial solution and then improve the solution by iterative algorithms. This technique is known as the heuristic method and can give a near-optimum solution within a practical runtime.

Most physical design algorithms are heuristic in nature. The quality of a heuristic algorithm can be directly evaluated, provided the optimal solution is known, In this case, its suboptimality ε is judged with respect to the optimal solution $\varepsilon = \frac{|S_H - S_{opt}|}{S_{opt}}$ where S_H is the cost of the heuristic solution and S_{opt} is the cost of the optimum solution. However, finding an optimum solution is impractical in most physical design problems. A typical approach in this scenario is to test heuristic algorithms against a suite of benchmarks. These benchmarks provide an assessment of the scalability of the algorithm and the possibility to compare the results with previously obtained solutions.

In this thesis, an iterative heuristic approach is used to improve the detailed routing score of a given optimized placement. Also, in this thesis, benchmarks from the ISPD 2018 and 2019 contests are used to evaluate the performance of the developed methods in the proposed framework.

3.8 Summary

In this chapter, the concepts used in this thesis are explained briefly. First, the concepts of complexity and graph theory are discussed. These are followed by the main data structure used for query processes in the development of the framework. Then, Dynamic programming techniques and Integer Linear Programming is illustrated to solve optimization problems. Problems and techniques to solve them are briefly presented. Finally, heuristic methods to solve optimization problems are given. All concepts presented in this chapter are used to develop the main contribution of this thesis.

Chapter 4

Proposed Framework

The proposed framework comprises three key components: a Global Routing (GR) engine, a Cooperation between Routing and Placement (CRP) engine, and a Detailed Routing (DR) engine. The focus of this chapter is to detail the development and evolution of the CRP engine. The CRP engine's steps are depicted in blue in Figure 4.1.

The CRP engine takes as input Library Exchange Format (LEF) and Design Exchange Format (DEF) files. The DEF file includes an initial legalized placement solution that has already been optimized. The Global Routing step uses CUGR [25] to generate an initial global routing solution based on the provided placement. Next, the CRP engine (depicted as the blue box in Figure 4.1) employs various techniques to integrate routing with cell movement to improve the routing quality. This process is applied iteratively until certain stopping criteria are met, which can be defined as a constant value k or a violation flag. The value of k represents the number of iterations and the violation flag indicates whether the routing solution contains any violations or not. The output of the CRP engine is a global routing and placement solution that is then provided to a detailed router (TritonRoute [91]) to perform physical wiring and VIA insertion. Finally, the results of the detailed routing and global routing are recorded as the output of the overall framework.

To determine the most appropriate global router and detailed router engines, a variety of experiments were conducted. The results of these experiments are summarized in Appendix B. The results of the experiments indicate that the CUGR is the optimal choice for a global router and TritonRoute is the optimal choice for a detailed router. Both of these engines have been found to perform better than other available routing engines.

Throughout the remainder of this chapter, I will first define the cost functions used in CRP in Section 4.1. Next, the five main steps of CRP will be thoroughly explained in Section 4.2. Lastly, I will provide an

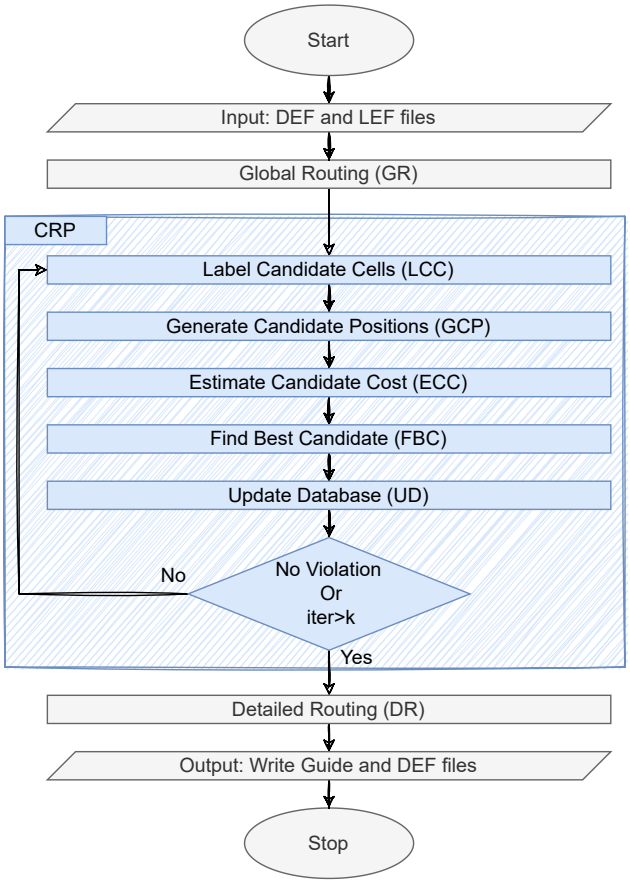


Figure 4.1: **An illustration of CRP Flow Framework**. The methodology flow incorporates three main steps. (1) Global Routing; (2) Cooperation between Routing & Placement (CRP); (3) Detailed Routing. The framework inputs are LEF and DEF files, and the outputs are DEF/Guide files.

illustration of the developed ILP-DP and extended global routing used by CRP in Section 4.3 and Section 4.4 respectively.

4.1 Cost Functions

To model the 3D grid graph of the global routing in CRP, I take inspiration from CUGR [25] to define the Capacity ($C(e)$), Demand ($D(e)$), edge cost ($cost(e)$), wire cost ($cost(wire)$), and VIA cost ($cost(via)$) as outlined in Definitions 4.1, 4.2, 4.3, 4.4, and 4.5, respectively. Furthermore, I use these parameters to calculate the Path Cost ($cost(path)$) as defined in Definition 4.6. The Path Cost is then used to evaluate the quality of each net routing.

Definition 4.1 (Capacity)**.** The capacity of an arbitrary edge $e(u, v)$, connecting G-Cell u to G-Cell v, refers to the total number of tracks available in the edge e. The capacity of an edge e is represented by $C(e)$.

For example, as illustrated in Figure 4.2, the capacity of the edge $e(u, v)$ is equal to 2, as there are two tracks available within the edge.

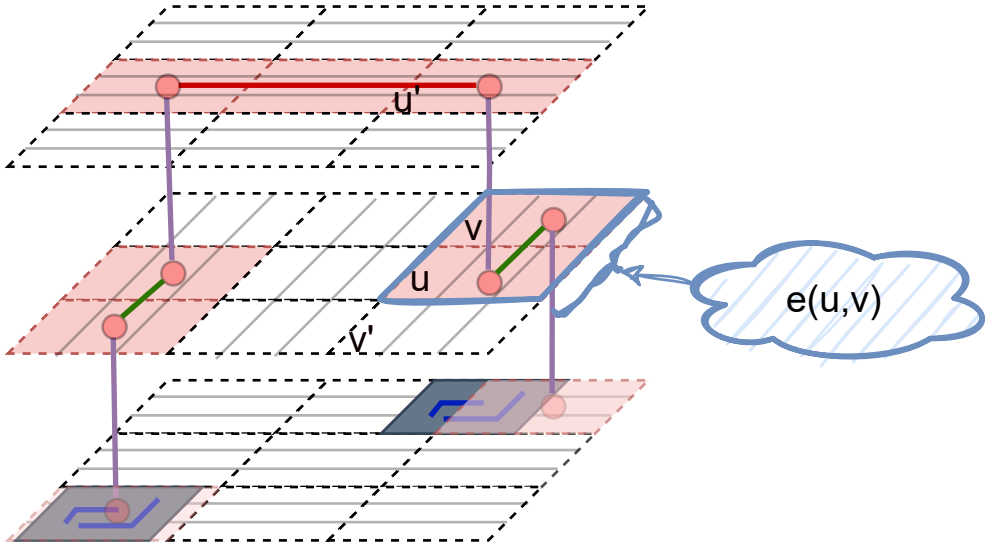


Figure 4.2: 3D global routing grid graph including edge and capacity modeling.

Definition 4.2 (Demand)**.** The demand of an edge e can be formulated as follows:

$$D(e) = U_w(e) + U_f(e) + \beta \times \delta(e) \tag{4.1}$$

Where $D(e)$ is defined as the sum of wire usage, represented as $U_w(e)$, passing through an edge e, the usage of fixed components, represented as $U_f(e)$, intersecting with edge e, and an approximation of the number of tracks affected by VIAs, represented as $\beta \times \delta(e)$. β is a constant factor, used in this work with the value of 1.5. $\delta(e)$ is modeled as $\sqrt{\frac{via(u)+via(v)}{2}}$ in CUGR [25], where $via(u)$ and $via(v)$ represent the number of VIAs in G-Cell u and v respectively. The same method for modeling $\delta(e)$ is also utilized in CRP to estimate the number of tracks affected by VIAs in an edge e. It should be noted that the number of VIAs in an arbitrary G-Cell u, represented as $via(u)$ includes both the VIAs from the lower and upper layers.

As an example of demand calculation in CRP, Figure 4.3 illustrates the demand calculation for an arbitrary edge $e(u, v)$. In this figure, the capacity of edge e is 7 as there are 7 available tracks within the edge, represented by dotted lines in the figure. The gray rectangle depicts a blockage. Additionally, there are two wires passing through the edge, wires 1 and 2, depicted as solid blue lines. Therefore, the demand of edge e ($D(e)$) equals $D(e) = 3 + 2 + 1.5 \times \sqrt{\frac{1+2}{2}} \approx 5.83$, With the assumption that the number of VIAs in u and v are 1 and 2 respectively. By utilizing this equation to calculate the demand for an edge, I obtain a less optimistic value for the number of available tracks in the edge. As illustrated in Figure 4.3, there are two

tracks available in the edge. However, our calculation suggests that only one track is practically available on this edge.

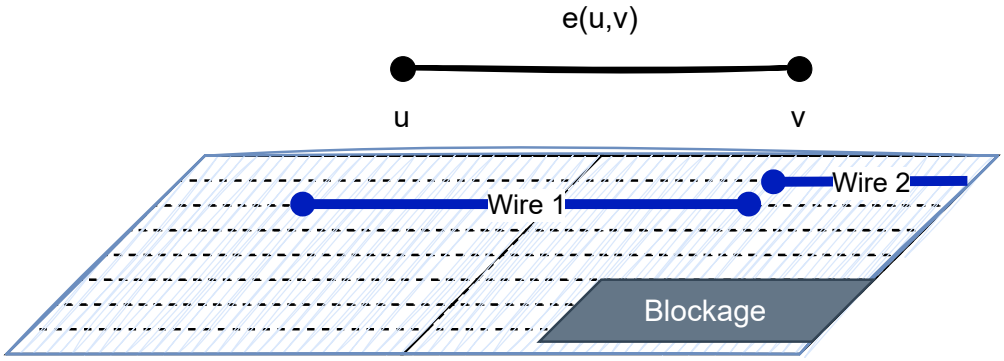


Figure 4.3: An example of the demand calculation of an arbitrary edge of $e(u, v)$.

Definition 4.3 (Edge Cost)**.** The cost of an edge e can be referred to as the cost or weight assigned to each edge of the routing grid during the global routing process. The cost of an edge can be calculated by considering both the length of the edge and the congestion of the edge [1]. The length of the edge can be calculated by using the Manhattan distance and the congestion function can be defined by a logistic function (as proposed in [114]). The cost of an edge is represented as $cost(e(u, v))$ or simply $cost(e)$. Therefore, the cost of an edge e can be defined as follows:

$$cost(e(u, v)) = MD(u, v) \times (1 + \text{penalty}(e(u, v))) \tag{4.2}$$

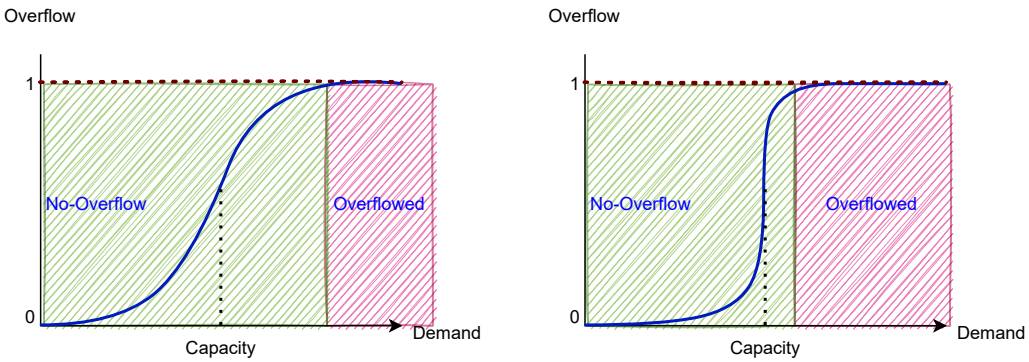
In equation (Equation 4.2), $MD(u, v)$ is defined as the Manhattan distance between the center of G-Cells, which is calculated as $|X(u) - X(v)| + |Y(u) - Y(v)|$. The $X(u)$ and $Y(u)$ represent the x and y coordinates of the center of G-Cells. The penalty for congested edges is composed of two parts: The cost of short violation for edges in each layer ($cost_{\text{short}}(L)$) and a logistic function ($\frac{1}{1+\exp^{(-S\times(D(e)-C(e)))}}$). This can be formulated as follows:

$$\text{penalty}(e(u, v)) = cost_{\text{short}}(e) \times (\frac{1}{1 + \exp^{(-S\times(D(e)-C(e)))}}) \tag{4.3}$$

The cost of a short violation of edges in each layer is represented by $cost_{\text{short}}(e)$, which is calculated as $cost_{short}(e) = (short_{length}(e) \times costUnit_{wire}(L(e))) \times costUnit_{short}(L(e))$, where $L(e)$ is a function that returns the layer that edge is assigned to, $short_{length}(e)$ represents the length of a short violation in edge e in DBU, $costUnit_{wire}(L(e))$ shows the cost of wiring per DBU in the layer that edge is assigned to, and $costUnit_{short}(L(e))$ shows the unit cost of a short violation in the layer. The $costUnit_{short}(L(e))$ is calculated

by $\frac{costUnit^{raw}_{short}(L(e))}{pitch(L(e))\times pitch(L(e))}$, where $costUnit^{raw}_{short}(L)$ is the short cost per unit area of $pitch(L(e)) \times pitch(L(e))$. The values for $costUnit^{raw}_{short}(l)$ and $costUnit_{wire}(l)$ where $l \in L$ is defined by ISPD 2018 and 2019 contests to represent the relative importance of each routing metric unit compared to others. They are 500 and 0.5 units respectively.

In the logistic function, $D(e)$ represents the demand, and $C(e)$ represents the capacity of the edge. The slope factor, S, can be adjusted to increase or decrease the sensitivity of the penalty for congested edges. As depicted in Figure 4.4, a change in the value S can lead to a change in the steepness of the curve. According to this figure, a higher value of S results in quicker overflow on an edge, while a lower value allows for more flexibility in the cost of edges. In the CRP engine, the value of S is set to 8.



(a) Logistic Function. Slope S equals to 1 (b) Logistic Function. Slope S equals to 8

Figure 4.4: The logistic function with different slope values. A higher slope value leads to a quicker approach of the logistic function to the value of 1, which can result in an overflow.

Definition 4.4 (Wire Cost)**.** The cost of a wire that connects two G-Cells can be defined as the summation of all edges in the wire as follows:

$$cost(wire) = \sum_{e \in E_{wire}} cost(e) \tag{4.4}$$

In Figure 4.5, a sample of a global routing wire is illustrated. The cost of the wire connecting G-cell u to G-cell v is computed by summing the costs of edges $e1$ and $e2$.

Definition 4.5 (VIA Cost)**.** The cost of inserting a VIA in the global routing between two G-Cells in adjacent layers can be calculated by summing the weight assigned to each VIA insertion and the congestion

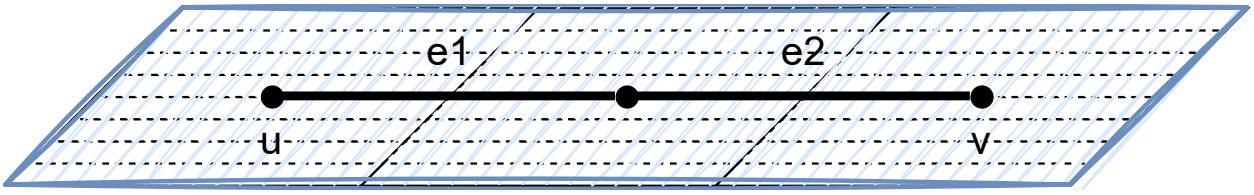


Figure 4.5: A sample of global routing wire that includes three G-Cells and is calculated by two edges.

of the edge as presented in [25]. Therefore, the cost of a VIA can be defined as follows:

$$cost(via(u \rightarrow u')) = costUnit_{via}(u \rightarrow u') \times (\gamma_{via} + penalty(via(u \rightarrow u'))) \tag{4.5}$$

In Equation 4.5, the unit cost of VIA insertion for G-Cell u to u' that are in adjacent layers is represented by $costUnit_{via}$(u $\rightarrow$ u'). The unit cost for a VIA insertion is set to 4 in this work, as defined by the ISPD 2018 and 2019 contests. The γ_{via} represents a unit VIA multiplier to weight each VIA insertion. By increasing the value of γ_{via}, the importance of VIA insertion can be increased. In the CRP engine, the value of γ_{via} is set to 4. Furthermore, the $penalty(via(u \rightarrow u'))$ represents the penalty for VIA insertion in congested areas and can be modeled as follows:

$$penalty(via(u \rightarrow u')) = \frac{1}{1 + exp(-S \times (D(u) - C(u)))} + \frac{1}{1 + exp(-S \times (D(u') - C(u')))} \tag{4.6}$$

In Equation 4.6, the penalty for VIA insertion between u and u' is computed by summing the logistic functions for G-Cell u and u'. According to the demand definition of each edge given in Definition 4.2, the demand of each G-Cell can be defined as the average of edge demands connected to G-Cell as given in Equation 4.7. The same formula is applied to G-Cell u'.

$$D(u) = \frac{1}{2} \times \sum_{e \in E_u} (U_w(e) + U_f(e)) \tag{4.7}$$

In Equation 4.7, the demand of a G-Cell u is calculated as the average of wire and fixed usage of connected edges to the G-Cell u (E_u).

As depicted in Figure 4.6, the cost of a VIA insertion in global routing, connecting u in the lower layer to u' in the upper layer, is dependent on the congestion of edges connected to each G-Cell. To calculate the congestion of G-Cell u, for example, the congestion in edges $e(v, u)$ and $e(u, w)$ are used.

Definition 4.6 (Path Cost)**.** The path cost of a routed net can be defined by the summation of the wire

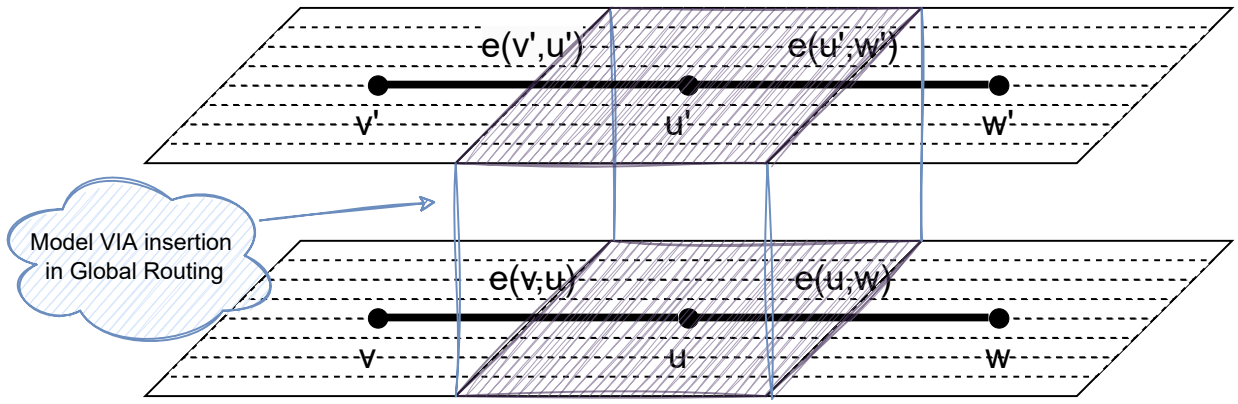


Figure 4.6: A global routing VIA insertion modeling sample consisting of two G-Cells u and u' in adjacent layers.

and VIA cost of the net as follows:

$$cost(path) = \sum_{wire \in Path_{wires}} cost(wire) + \sum_{via \in Path_{vias}} cost(via) \tag{4.8}$$

The path cost defined in Equation 4.8 is utilized to evaluate the quality of each placement. As stated in the problem formulation in Equation 2.1, this cost is used by the CRP algorithm to determine the quality of each placement. A sample routed path connecting pin Y of inst1 to pin A of inst2 is illustrated in Figure 4.7. The path incorporates two VIAs (via1 and via2) and a single wire (wire1), and the cost of this path is calculated based on the costs of via1, via2, and wire1.

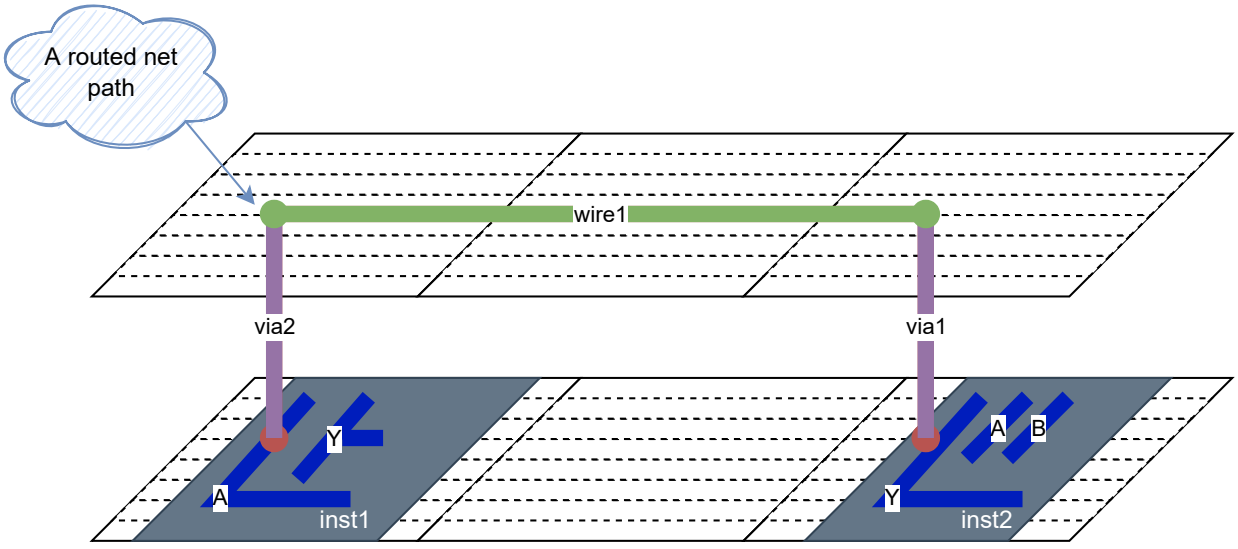


Figure 4.7: A sample of a routed path that connects the pin Y inst1 to pin A in inst2 is given. The path incorporates two VIAs (via1 and via2) and a single wire (wire1).

4.2 Cooperation between Routing and Placement (CRP) Engine in Advanced Technology Nodes

As shown in Figure 4.8, the CRP engine aims to improve cell positioning and overall routing quality by incorporating five main steps: Label Candidate Cells (LCC), Generate Candidate Positions (GCP), Estimate Candidate Cost (ECC), Find Best Candidates (FBC), and Update Database (UD). In LCC, candidate cells are selected based on their initial routing path cost. GCP uses an ILP-DP engine to propose new candidate locations that meet technology constraints. ECC evaluates the quality of possible movements using 3D global routing and employs caching techniques to improve runtime. FBC selects the best movement for candidate cells, and UD updates the database by moving the selected cells. The techniques used in each step will be described in further detail in this section.

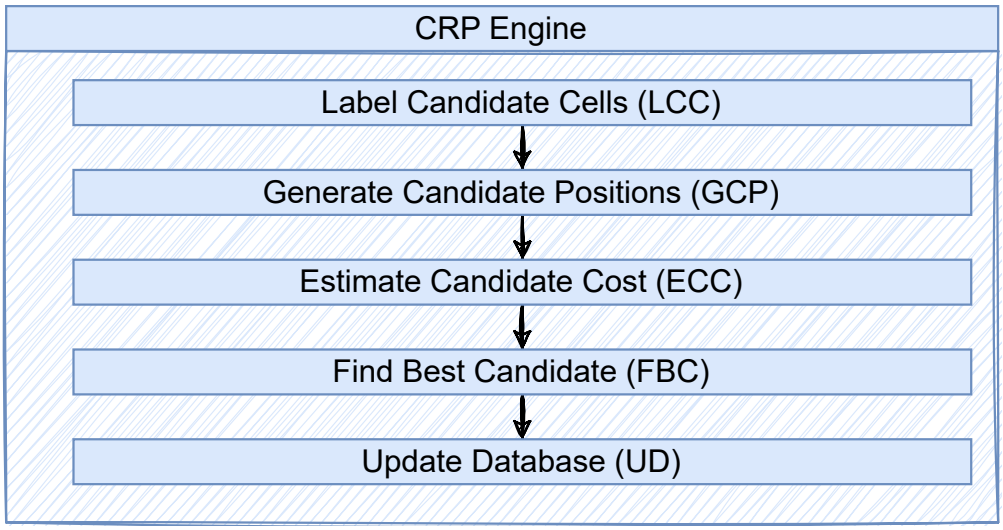


Figure 4.8: **An illustration of CRP Engine**. The CRP engine incorporates five main steps. (1) Label Candidate Cells (LCC); (2) Generate Candidate Positions (GCP); (3) Estimate Candidate Cost (ECC); (4) Find Best Candidates (FBC); (5) Update Database (UD).

4.2.1 Label Candidate Cells (LCC)

The philosophy behind candidate cells is to identify cells with the highest cost location and move them to a new position to reduce the total cost of all cell positions. The cost function used to evaluate the position of each cell is defined in Section 4.1, and it is calculated by 3D routing of nets connected to each cell. However, moving all cells and calculating 3D routing for each movement is a computationally expensive task. To address this, a percentage of cells is designated as input variable.

In this work, 0.6 percent of cells are selected in each iteration. This way, cells with the highest cost

position are targeted over iterations. By selectively choosing which cells to move, the algorithm can reduce the total cost over iterations. Overall, the philosophy of candidate cells is to strike a balance between exploring the search space for potentially better solutions and exploiting the current best solutions. By iteratively selecting and moving cells with the highest cost positions, the algorithm can converge on a better overall solution through iteration. In this work, the number of iterations is set to 10 iterations for the proof of concept that the framework can improve the detailed routing result after movement. The algorithm and its proposed steps are described in detail below.

Algorithm 1 illustrates the process by which the CRP engine selects a set of candidate cells to move in each iteration. The algorithm first calculates the path cost of nets connected to each cell and the median of each cell (line 1), inspired by [64]. It then copies a set of cells from the database (db) into the set C (line 2), and sorts this set based on the path cost of initial global routing for the nets of each cell (line 4). Cells with a higher path cost of nets will be ordered at the top of set C.

Next, the algorithm loops over each cell in the set C (line 5), and investigates each cell according to three main factors: (1) no cell connected to cell c should be found in the candidate cell set (line 6); (2) if cell c was already a candidate cell in earlier iterations of the flow, the value of $hist_c$ (history of candidate cell c) will be 1 (line 10); and (3) if cell c was already moved in earlier iterations of the flow, the value of $hist_m$ (history of moving candidate cell c) will be 1 (line 11). Factors (2) and (3) are both used to reduce the probability of selecting cells that were affected in earlier iterations of CRP.

A probability of accepting cells to move, inspired by [54], is applied for the selection of each cell (line 12). For example, if a cell was only selected as a candidate cell in earlier iterations (not moved), the probability of selecting the cell again in the current iteration is 36% (exp(-1)). If the cell was moved in earlier iterations, this probability reduces to 13% (exp(-2)). This approach helps the framework to address more cells to move and avoid getting stuck with candidate cells in congested areas. Finally, in line 16, the value of γ (set to 0.6 in this work) tells the framework what percentage of cells the flow wants to move. The selected candidate cells are then returned by the LCC function.

An example of LCC is presented in Figure 4.8. The example shows a set of cells that are connected to each other by three nets (N1, N2, and N3). The example provided shows that in the initial state (shown in Figure 4.8(a)), the cell "c1" has the highest path cost among all cells and it is connected to two nets (N1 and N2). As a result, LCC algorithm selects "c1" as a critical cell and all the cells connected to it are fixed (as shown in Figure 4.8(b)). For other cells close to c1, the LCC algorithm selects randomly a cell for a candidate movable cell. For example, the cell "c8" is considered a movable cell, and "c9" is fixed.

Algorithm 1: Label Candidate Cells (LCC) to move

Input: A set of cells C from database (db).
Output: A set of candidate cells $candidate_{set}$ to move.

updateCellFeatures(db.cells); `// Calculate an initial path cost (Definition 4.6) and median of each cell [64]`
C = copy(db.cells); `// A set of cells from database (db)`
$candidate_{set} = \{\}$; `// A set of candidate cells to move`
sort(C) ; `// Sort the set of Cells according to Path Cost Definition 4.6`
foreach c *in* C **do**
    $connected_{cells}$ = c.getConnectedCells();
    if $candidate_{set}.find(connected_{cells})$ **then**
        continue;
    end
    $hist_c$ = db.$candidate_{set}^{hist}$.find(c);
    $hist_m$ = db.$moved_{set}$.find(c);
    probability_of_accepting = exp(-1*($hist_c$+$hist_m$)) / T ;
    if $probability\_of\_accepting > random(0, 1)$ **then**
        $candidate_{set}$.insert(c);
    end
    if $len(candidate_{set}) > \gamma * len(C)$ **then**
        break ;
    end
end
return $candidate_{set}$

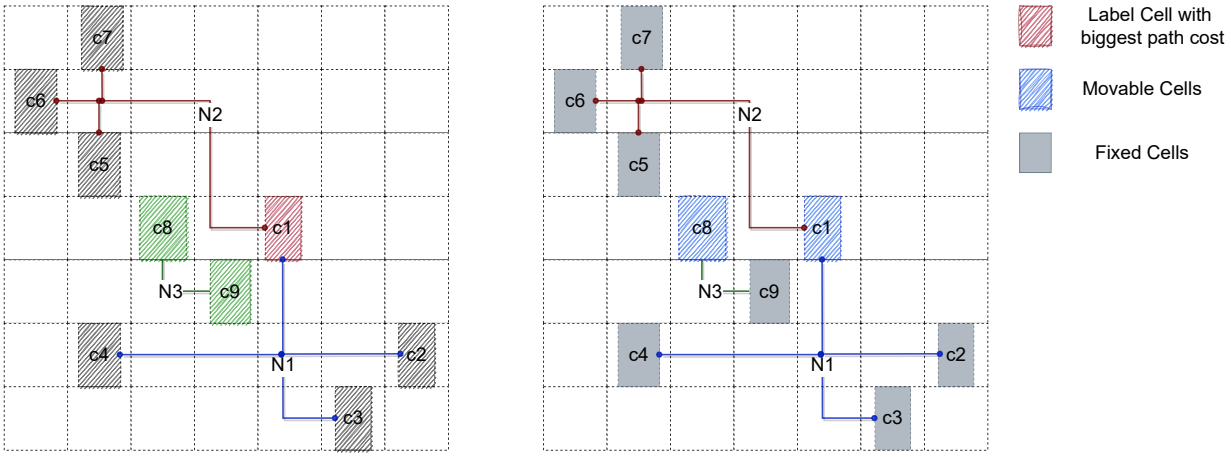


Figure 4.8: A sample of labeling critical cells with the biggest path cost.

4.2.2 Generate Candidate Positions (GCP)

The process of generating placement candidates is detailed in Algorithm 2. A placement candidate is defined as a pair of coordinates, (x,y), which represent the bottom-left corner of a potential cell position. The objective of the GCP step is to identify and generate legal and valid positions for candidate cells, which were

previously tagged in the LCC step (Algorithm 1) and stored in *candidate_set*.

Algorithm 2 consists of two main loops. The first loop, from line 1 to 6, is used to find placement candidates for each candidate cell and operates in parallel. As the newly generated placement candidates may cause conflict with the initial positions of other cells, these cells are referred to as conflict cells. To resolve these conflicts, the second loop, from line 7 to 13, is implemented to add new placement candidates for conflict cells in a sequential manner. At the end, all cells involved in the movement of candidate cells and their placement candidates are added to the movable set ($movable_{set}$), which is then returned by the function.

A walk-through example is provided for Algorithm 2 to enhance understanding of the GCP step. In line 2, each cell gets its initial placement position as one of the placement candidates. This means that in the worst-case scenario the cells will keep their original positions. An ILP-based Detailed Placement (ILP-DP) is proposed to find a legalized solution for all cells after a movement in a local area. In line 3, the ILP-DP receives three inputs: a candidate cell c, a number of sites (N_{site}), and a number of rows (N_{rows}) around the cell c. The ILP-DP will return a pair set of placement candidates for the candidate cell c (*placement_candidates*), and a list of conflict cells with their new legalized positions (conflict_cells). This is shown in Figure 4.9. In this figure, c1 is selected from the $candidate_{set}$ to move, with c2 and c3 as movable cells and c4 and c5 as fixed cells. The initial placement for c1, c2, and c3 is shown in Figure 4.9 (a). If c1 moves from (row2, site0) to (row1, site1), as illustrated in Figure 4.9 (b), it will result in overlap with c2. To address this overlap, ILP-DP creates new placement options for c2 that are legalized and avoid an overlap with c1, and stores them in the $conflict\_cells$ variable. For instance, one possible new placement option for c2 is (row2, site0), which does not overlap with c1, as shown in Figure 4.9 (c). Now, these placement options for cells c1 and c2 must be added to the vector of *placement_candidates* for each cell. Therefore, the placement options and the conflicting cells of c1 are pushed back to variables *placement_candidates* and $conflict\_cells$ in line 4-5 of Algorithm 2. Next, the $conflict\_cells$ new positions with cell c1 will sequentially be pushed as a new candidate position for the conflict cells in line 7-13. The reason for pushing $conflict\_cells$ sequentially is that they may have conflicts with other labeled cells in another window, leading to conflicts during parallel processing. line 8 to 11 of the algorithm loop over the conflict cells of c1. Within the loop, c2, as a conflict cell of c1, will receive new placement options and will be added to the *movable_set* in line 10. Then, in line 12, c1 will also be added to the *movable_set*. Finally, the *movable_set* will return by the function. In Figure 4.9, the *movable_set* includes cells c1 and c2. The placement candidates for c1 are (row2,site0) and (row1,site1), and for c2 are (row1,site0) and (row2,site0). This shows for any position for candidates' cells to move, there can be multiple legalized solutions. Also, pushing the new positions for the conflict cells of a candidate cell will guarantee for each movement at least one legalized solution for all the

cells in the circuit exists.

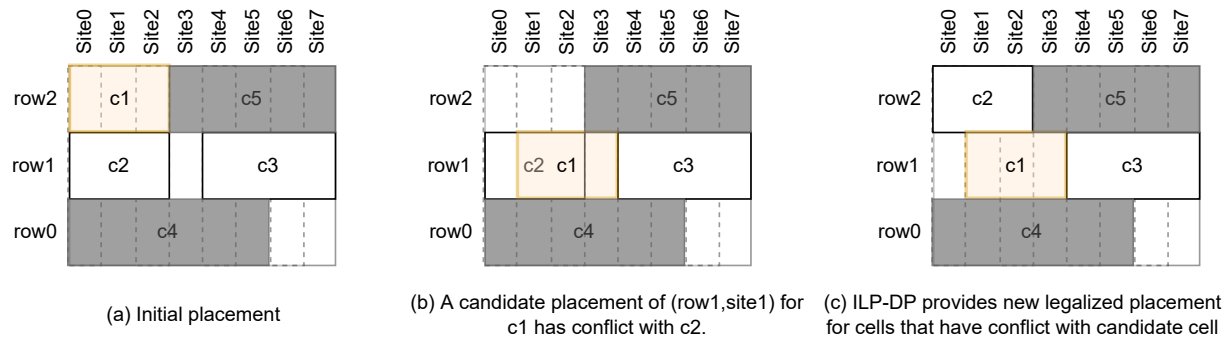


Figure 4.9: ILP-DP in Algorithm 2 can find a new legalized solution for a candidate cell. In this scenario, c1 is a cell in $candidate_{set}$ to move and in the given window c2 and c3 are movable cells and c4 and c5 are fixed cells. (a) Initial placement for c1, c2, and c3 (b) c1 moves to (row1,site1) has conflict with c2. (C) An alternative overlap-free solution.

Algorithm 2: Generate Candidate Position (GCP)

Input: A list of candidate cells to move $candidate_{set}$
Output: A set of movable cells with multiple candidate position ($movable_{set}$)
`// run parallel`
foreach *c in* $candidate_{set}$ **do**
  $c.placement\_candidates$.add(getCurrentPos(c));
  ($placement\_candidates$, $conflict\_cells$) = ILP-DP.run(c,N_{site},N_{row}); `// Section 4.3`
  $c.placement\_candidates$.add($placement\_candidates$);
  $c.conflict\_cells = conflict\_cells$;
end
`// run sequential`
foreach *c in* $candidate_{set}$ **do**
  foreach ($c^{'}$, $placement\_candidates$) *in* $c.conflict\_cells$ **do**
    $c^{'}.placement\_candidates$.add($placement\_candidates$);
    $movable_{set}$.push($c^{'}$)
  end
  $movable_{set}$.push(c)
end
return $movable_{set}$

4.2.3 Estimate Candidate Cost (ECC)

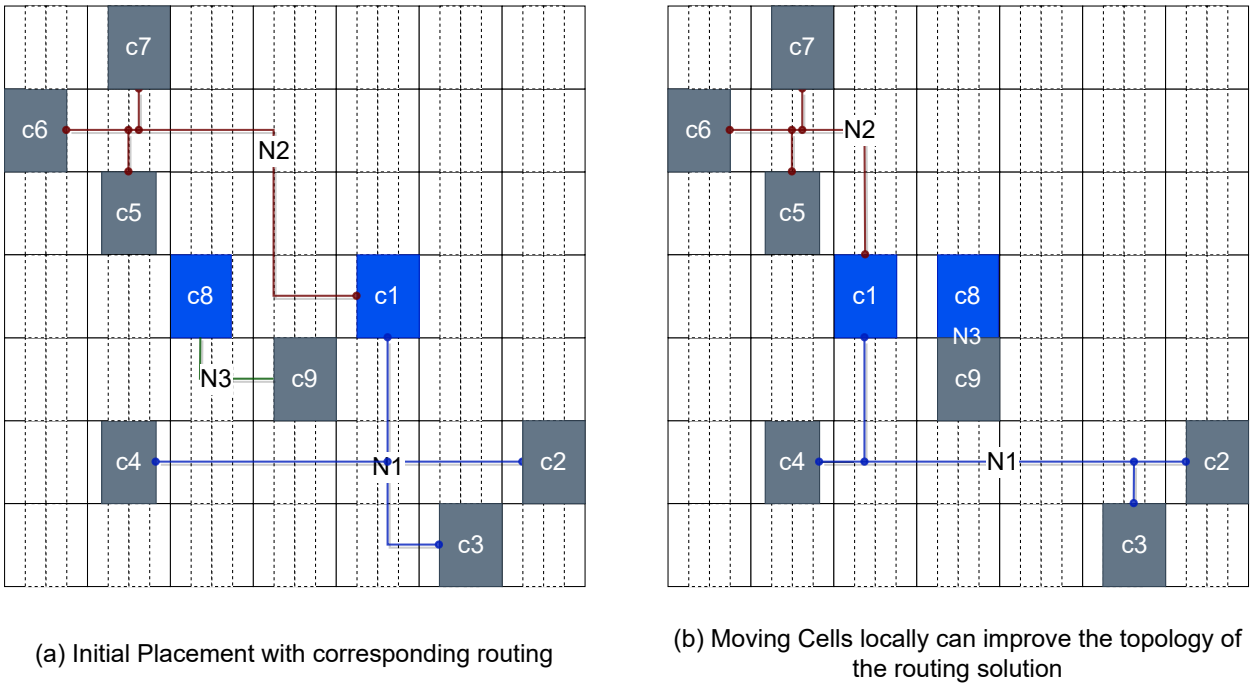
After generating possible placement positions for each candidate cell and their conflict cells, it is necessary to evaluate the quality of their new possible positions. This quality is determined by the proposed path cost function (Equation 4.8). As shown in Algorithm 3 and line 2, for each movable cell, the connected nets are queried. Then, based on the class of the net (determined in line 5), each net is either routed by the PatternRoute (line 8) or AStar (line 10) algorithm. A class of nets that shows the required algorithm for routing the net, there are two classes of nets available in CRP: PATTERNROUTE and AStar. Within each class, CRP can define the specific algorithm that should be used for routing the net. For example, the PATTERNROUTE class uses a PatternRoute algorithm, while the AStar class uses the AStar pathfinding algorithm. To determine which class to use for routing a particular net, CRP includes logic in your code to evaluate the characteristics of the net and determine the most appropriate algorithm to use. The class net determination as well as the PatternRoute and AStar routers used in CRP are described in detail in Section 4.4. The path cost of the 3D route is used to calculate the cost of the new position of the candidate cell (line 13). If the routed net has any violations (line 12), the candidate position for the movable cell c will be removed from the placement candidates in line 15. Note that multiple cell movements are allowed per iteration, but only one cell per net is permitted to move to ensure the estimated routing of its connected cells remains valid. The other connected cells to a moved cell stay fixed.

Algorithm 3: Estimate Candidate Cost (ECC))

Input: A set of movable cells with multiple candidate position ($movable_{set}$)
Output: Estimating a cost value for placement candidates of each candidate cell

```
1  foreach c in movable_set do
2      N = getNetsConnectedToCell(c);
3      foreach n in N do
4          C_n = getCellsConnectedToNet(n);
5          Net_class = ClassifyNet(n);                                  // Section 4.4
6          foreach placement_candidate in c.placement_candidates do
7              if Net_class == PATTERNROUTE then
8                  n_routed3D = getPatternRoute3D(C_n,placement_candidate);
9              else
10                 n_routed3D = getAStar3D(C_n,placement_candidate);
11             end
12             if hasNoViolation(n_routed3D) then
13                 placement_candidate.cost += n_routed3D.getCost();
14             else
15                 removeCandidate(placement_candidate,placement_candidates);
16             end
17         end
18     end
19 end
```

An example of ECC after cell movement is depicted in Figure 4.10. Although the routing of each placement is done in a 3D grid graph of the global routing in the CRP engine, for simplicity, a 2D view of routing is shown in this figure. After each placement, the connected nets to the moved cells are rerouted by global routing. In this example, after moving cells $C1$ and $C8$, the nets $N1$, $N2$, and $N3$ are rerouted. The calculation of the total routing path provides information about the quality of the placement and their corresponding routing. As can be seen in this example, Figure 4.10(b) has a better routing solution with fewer detours and wirelength.



(a) Initial Placement with corresponding routing

(b) Moving Cells locally can improve the topology of the routing solution

Figure 4.10: A 2D view of two sample placements, along with their corresponding routing paths.

4.2.4 Find Best Candidate (FBC)

To select the best candidate placement among all possible candidates for moved cells, the FBC step is developed in the CRP. As explained in earlier chapters, a cost value is calculated for each candidate placement (as outlined in Algorithm 3), and this cost value defines the $cost_c^p$ for each placement of cell c in Equation 2.1.

To implement Equation 2.1 and the legalization constraints for each placement (Equation 2.2-Equation 2.7), the FBC step incorporates two main phases: Construct Constraint Graph (CCG) and Solve ILP Model (S-ILP). In the following, the CCG and S-ILP steps are illustrated in detail.

4.2.4.1 Construct Constraint Graph (CCG)

There are two types of constraints that must be satisfied to have a valid (legal) detailed placement solution. First, for each cell, only a single position can be selected as a solution (Equation 2.2, line 10 in Algorithm 4). Second, the solution must guarantee that there is no overlap between the cells (Equation 2.5, line 25 in Algorithm 4). In Figure 4.11, the construction of the single position and overlap constraint graphs is shown.

In Figure 4.11(a), a segment with two movable cells (c1 and c2) and six placement positions (p1,p2,p3,p4,p5,p6) is given. The cost of placing cells c1 and c2 can be stored in a weights vector as shown in Figure 4.11(b). In the weights vector, the first five indexes (index 1-5) are assigned to the cost of placing c1 in available positions. The rest of the costs (index 6-9) are assigned to the cost of placing c2 in each position. The indices in the weights vector can be used as the nodes of the constraint graphs, as shown in Figure 4.11(c). In Figure 4.11(c), the unique position of the constraint graph represents that for each cell one position is allowed to be selected (black edges). The red edges represent the overlap constraints between cells. For example, if index 1 in the weights vector is selected for cell c1, this means that c1 is placed in p1. Therefore, c2 can not be placed in p1 (index 6) and p2 (index 7), otherwise, c1 and c2 will overlap. Finding an independent set of this constraint graph results in a valid and legalized placement, as shown in Figure 4.11(d).

To find the constraint graph, four loops are developed as shown in Algorithm 4. In the first loop (line 3-line 7), the index of each cell in the movable set ($movable_{set}$) along with its possible candidate position and the cost of the position will be pushed into a list called weights. The second loop (line 8-line 11) pushes the constraint that only one position is allowed for each candidate cell to the constraints list. To find the overlap between the candidate cells' positions, an R-tree [101] data structure is used. In the third loop (line 13-line 17), the R-tree is filled according to the candidate geometry. In R-tree, each entry is an index and the cell's rectangle. Next, in the fourth loop (line 19-line 29), the overlapped cells will be queried from the R-tree. To avoid duplication in the constraint graph, a constraint map (line 18) that stores the combination of overlapped candidate indexes is used. Therefore, before adding each overlap constraint, in line 24, it is

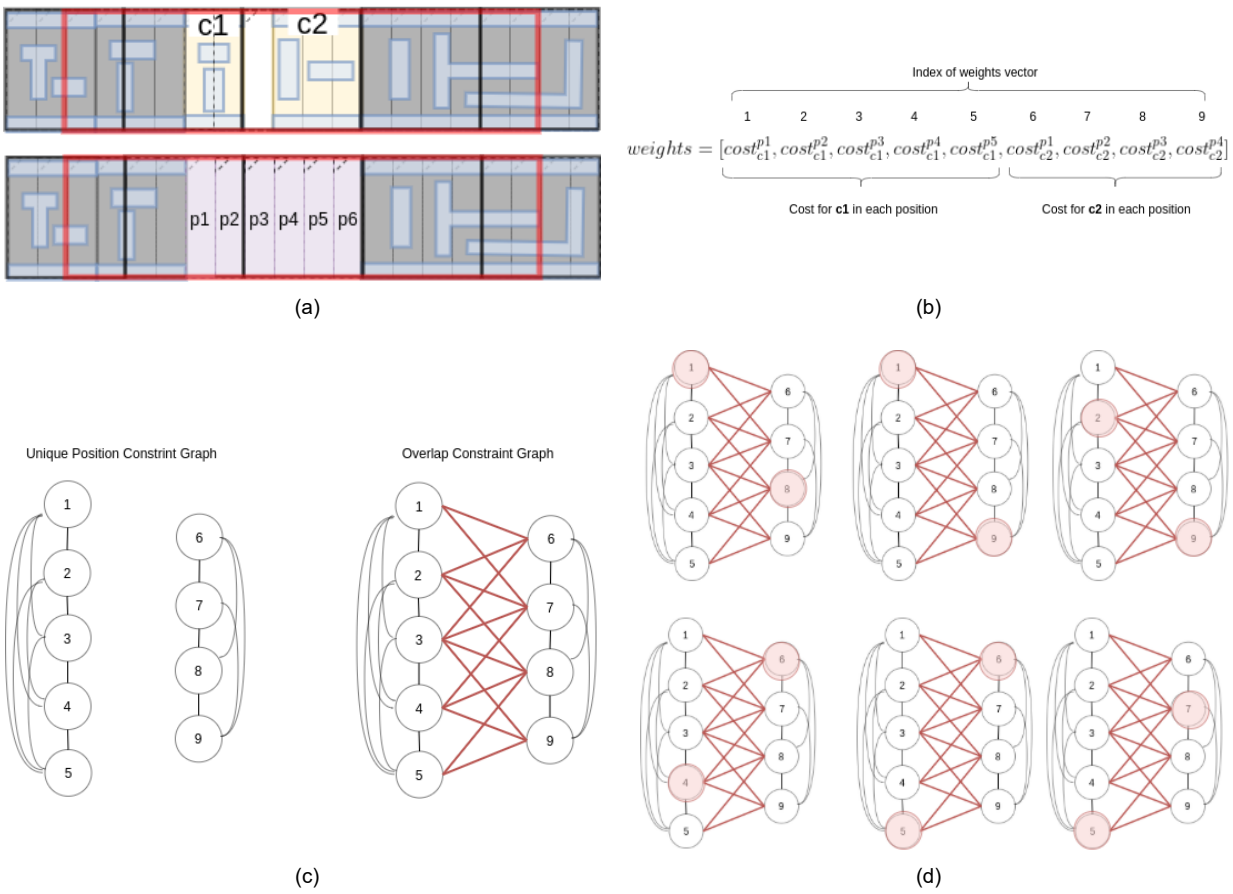


Figure 4.11: Illustrating the construction of a Constraint Graph based on a given segment with 2 movable cells and 6 placement sites. (a) Shows two movable cells (c1 and c2) and 6 available sites (p1,p2,p3,p4,p5,p6). (b) shows the vector of weights. Each item of the vector represents the cost of cell c in position p. (c) shows the unique and overlap constraint graph. Each edge represents a conflict between indexes of the weights vector. (d) possible independent set solutions for the conflict graph.

investigated if the overlap is already in the constraint graph or not, then it will be pushed into the constraint graph. Finally, the weights and constraints list will be returned by the CCG function. These two variables will be used to construct an ILP model, and solving this ILP model allows us to generate a new legalized placement while optimizing the path cost.

4.2.4.2 Solving ILP (S-ILP)

After constructing the costs of each placement candidate and the overlap conflict between these candidates, to find the possible solution, these costs and constraints are converted to an ILP model. As shown in Algorithm 5, the objective and constraint expressions are initialized in line 1 and line 2. Then, in line 3, a model is instantiated using CPlex [110] as the solver. In line 4, a list of variables with the size of the weights vector is generated, where each variable i in the vars set represents a respective candidate position and it can

Algorithm 4: Construct Constraint Graph (CCG)

Input: A list of movable cells ($movable_{set}$)
Output: A list of cell index and candidate positions (weights). A list of constraints including the conflict between weights list indexes.

```
1  weights={};
2  constraints.resize(movable_set);
3  foreach c in movable_set do
4  |   foreach pl_cd in c.placemnt_candidates do
5  |   |   weights.push((c.idx, pl_cd, pl_cd.cost));
6  |   end
7  end
8  foreach i in weights.size() do
9  |   w = weights[i];
10 |   constraints_uniq[mapIdx(w.idx)].push(i);          // Single position for each cell
   |    (Equation 2.2)
11 end
12 rtree=constructRTree();
13 foreach i in weights.size() do
14 |   w = weights[i];
15 |   cand_box= getCandidateBox(w);
16 |   rtree.insert(cand_box,i);
17 end
18 const_map = {};
19 foreach i in weights.size() do
20 |   w = weights[i];
21 |   q_box = getQueryBox(w);
22 |   q_res = rtree.query(q_box);
23 |   foreach j in q_res.size() do
24 |   |   if !(const_map.find(pair(i, j))) then
25 |   |   |   constraints_ovrlp.push(pair(i,j));      // Overlap cells positions (Equation 2.5)
26 |   |   |   const_map.insert(pair(i,j));
27 |   |   end
28 |   end
29 end
30 return [weights,constraints_uniq,constraints_ovrlp]
```

be either 0 or 1. If variable i is zero it means that this candidate position is not selected, and 1 means that the candidate position is the selected position for a movable cell. In the first loop (line 6-line 8), the cost of each candidate is used to construct the objective expression. Then, two types of constraints, including selecting one position for each cell and no overlap between cells, are generated in the second (line 9-line 15) and third (line 16-line 22) loops. In line 23, the objective expression (expr_obj) is added to the model to be minimized and the constraint expressions are added to the model in line 24. The model is then solved in line 25 which returns a new optimized placement. Finally, the new placement results are returned by the function (line 26) to be applied to the main database.

Algorithm 5: Solving ILP (S-ILP)

Input: *weights*,*constraints_uniq*,*constraints_ovrlp*
Output: A new placement solution (placement_sols)

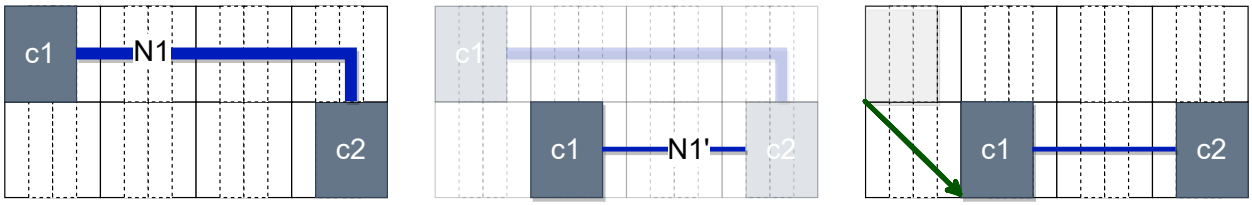
```
1  expr_obj={};
2  constraints_expr={};
3  model = getModel();
4  vars= constructVarsExprs(len(costs));
5  i = 0;
6  foreach w in weights do
7  |  expr_obj += w.cost × vars[i] ;
8  end
9  foreach confs in constraints_uniq do
10 |  expr_con={};
11 |  foreach conf in confs do
12 |  |  expr_con += conf ;
13 |  end
14 |  constraints_expr.add(expr_con == 1) ;
15 end
16 foreach confs in constraints_ovrlp do
17 |  expr_con={};
18 |  foreach conf in confs do
19 |  |  expr_con += conf ;
20 |  end
21 |  constraints_expr.add(expr_con ≤ 1) ;
22 end
23 model.add(Minimize(expr_obj)) ;
24 model.add(constraints_expr) ;
25 placement_sols = model.solve();          // The CPlex is used to solve ILP problem.
26 return placement_sols;
```

4.2.5 Update Database (UD)

After identifying the new positions for the candidate cells (*placement_sols* in Algorithm 5), they are physically moved to those positions, and the database is updated accordingly. This includes rerouting the nets connected to the moved cells using the global router (either PatternRoute or AStar, depending on the net's class), as well as updating the congestion and blockage maps in the database. Once the database has been updated, the new global routing solution and the updated cell positions can be passed to the detailed router for further improvement, or the process can be repeated for additional improvement.

To illustrate the process of the UD step, a sample of updating the database is provided in Figure 4.12. In this example, it is assumed that two cells C1 and C2 are connected by N1, Figure 4.12(a). During the GCP and ECC stages, the main database remains unchanged. As part of the GCP, cell C1 is virtually moved and routed to a new position as shown in Figure 4.12(b). If, during the FBC step, it is determined that this movement will result in an overall improvement in routing quality, the cell is then physically moved to the new position in the UD step, as shown in Figure 4.12(c).

The decision to separate the main database from the evaluation of each movement is driven by two key factors. Firstly, there is a possibility that routing results may deteriorate after applying a movement in a subsequent iteration, as the CRP engine operates in a greedy approach and only applies movements that result in an improvement in the routing cost. As such, movements that are unlikely to enhance the overall routing solution will not be applied to the database. Secondly, separating the main database from movement evaluation allows for the implementation of multi-threading during the CRP flow, leading to significant improvements in runtime performance.



(a) Initial Placement with corresponding routing

(b) To evaluate the routing quality during ECC, virtually move cell C1 to a new position.

(c) If the quality of N1' is superior to that of N1, the cell is physically relocated to the new position in the UD step.

Figure 4.12: A sample to illustrate the Update Database (UD) step in CRP

4.3 ILP-based Detailed Placement (ILP-DP) in CRP

In this section, the detailed placement used in CRP is explained in detail. This function is called in line 3 of Algorithm 2. The main objective of ILP-DP is to generate a new legalized solution for each candidate cell and its conflict cells. In order to cooperate the detailed placement with the global routing, there are two main challenges: the definition of a cost function that guarantees the quality of candidate positions, and the runtime of the detailed placement. Therefore, to cope with the above-mentioned challenges, the mathematical model and the proposed cost functions of ILP-DP are illustrated in Subsection 4.3.1. Then, the main steps of ILP-DP shown in Figure 4.13 are discussed in Subsection 4.3.2.

4.3.1 ILP-DP Model

The ILP-DP can be modeled as follows:

$$\min \sum_{c \in C} \sum_{j=1}^{N_{row}} \sum_{i=1}^{N_{site}} cost_c(i,j) \times y_c^{(i,j)} \tag{4.9}$$

In Equation 4.9, $cost_c(i,j)$ represents the cost of cell c in site i and row j. The value of $y_c^{(i,j)}$ is 1 if cell c is located in site i and row j, and 0 otherwise. One of the main challenges in this model is evaluating the quality of candidate positions. The problem is that the detailed placement can generate a large number of low-quality candidate positions, which adds significant overhead to the flow. Investigating the quality of each candidate position using global routing is a time-consuming task, especially for cells with global nets. To address this problem, a Congestion-aware Moving-Toward Cell's Median (CM-TCM) cost function is proposed. This cost function is inspired by the median idea of [115] and aims to identify high-quality choices for candidate positions. The CM-TCM cost function is defined such that an optimal region for a cell c is the median of cell c while the other connected cells to cell c are fixed. Therefore, a possible good placement candidate is a placement where the distance summation of all cells to their median in a given window is minimized.

Definition 4.7 (CM-TCM)**.** The cost of a possible candidate location ($cost_c(i,j)$) for cell c is proposed as follows:

$$cost_c(i,j) = dist_{c_{med}}(i,j) \times (1 + \text{penalty}(c,(i,j))) \tag{4.10}$$

In Equation 4.10, $dist_{c_{med}}(i,j)$ defines the distance between the current location of cell c and the median of the pins connected to it, and it is computed as follows:

$$\begin{aligned} dist_{c_{med}}(i,j) = W(site) \times |X(c,(i,j)) - X(c,(i_{\text{median}}, j_{\text{median}}))| + \\ H(row) \times |Y(c,(i,j)) - Y(c,(i_{\text{median}}, j_{\text{median}}))| \end{aligned} \tag{4.11}$$

Where $X(c,(i,j))$ and $Y(c,(i,j))$ are the x and y coordinates of the current location of the cell c, and $X(c,(i_{\text{median}}, j_{\text{median}}))$ and $Y(c,(i_{\text{median}}, j_{\text{median}}))$ are the x and y coordinates of the median of the pins connected to it, and $W(site)$ and $H(row)$ are the width and height of the site and row respectively. The $W(site)$ and $H(row)$ in Manhattan distance are used because the height and width of the sites and rows are not equal. The penalty$(c,(i,j))$ represents the penalty of cell c in the location of the site i and row j, and it is proposed as follows:

$$\text{penalty}(c,(i,j)) = \frac{\Gamma}{1 + \exp^{(\text{-S} \times (c_{\text{fan-out}} + D(i,j) - C(i,j)))}} \tag{4.12}$$

In Equation 4.12, $c_{\text{fan-out}}$ represents the number of connections to the cell. The $D(i,j)$ shows the demand for

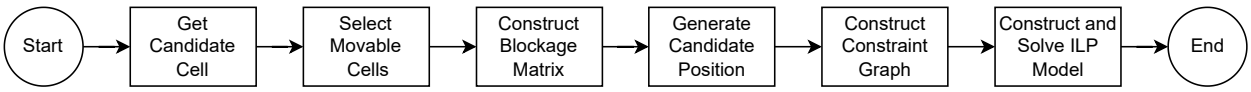


Figure 4.13: This Figure shows six steps in the Integer Linear Programming Detailed Placement (ILP-DP) Flow.

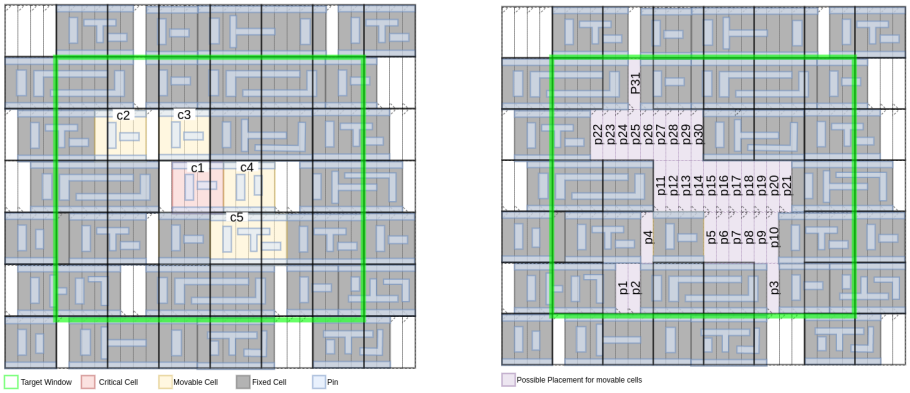
the G-Cell in the location of (i,j), calculated by Equation 4.1. The $C(i,j)$ shows the capacity of a G-Cell in the location (i,j). Moreover, to avoid placement under special nets (like VDD/GND) a penalty factor called Γ is defined in Equation 4.12. If there is an overlap between the cell and upper layer special nets, then Γ is a big number otherwise it is 1 [1]. This penalty is defined to avoid congested areas and placement under special nets in generating candidate positions for each cell.

4.3.2 ILP-DP Flow

As shown in Figure 4.13, the detailed placement incorporates six main steps. First, the candidate cell within the size of a window is given to the detailed placement. Then, movable cells are selected inside the detailed placement window along with the candidate cell, in *"Select Movable Cells"*. After that, all fixed cells are considered soft blockages, and a blockage matrix is generated according to the fixed cells in *"Construct Blockage Matrix"*. Then, different positions for movable cells as possible placements will be generated in "Generate Candidate Positions" step. The cost of each candidate position inside the window is calculated by the proposed CM-TCM cost function (explained in Subsection 4.3.1). Next, according to the positions of all movable cells, a conflict matrix guarantees an overlap-free placement with a single position for each cell generated in the *"Construct Constraint Graph"* step. For Construct Constraint Graph (CCG), Algorithm 4 is reused. Finally, the cost of each position, together with the constructed conflict matrix, is used to build an ILP problem. Solving this ILP problem can provide a new placement for a given area (in *"Construct and Solve ILP Model"*). For solving the ILP model in detailed placement, Algorithm 5 is reused. Later, the quality of this placement can be investigated by the global routing in order to apply to the main database (explained in Subsection 4.2.3).

In the ILP-DP flow, the "Generating Candidate Position" is the most time-consuming step. This is shown in Figure 4.14 and the techniques to speedup the ILP model in generating candidate positions are presented in Figure 4.15. In Figure 4.14(a), it is assumed that there are five movable cells ($movable_{set} = \{c1, c2, c3, c4, c5\}$). In the given window, there are 31 empty sites, shown in Figure 4.14(b). Each empty site is the potential position for each cell to be placed. Therefore, the time complexity of placing m movable cells ($|movable_{set}|$) in n available sites ($|sites|$) is equal to $\frac{n!}{(n-m)!}$. This complexity even for a small window size

[1] Γ can be an infinite number if placement sites under a special net are considered as a soft blockage.



(a) ILP-DP window with 4 movable cells and 1 critical cell

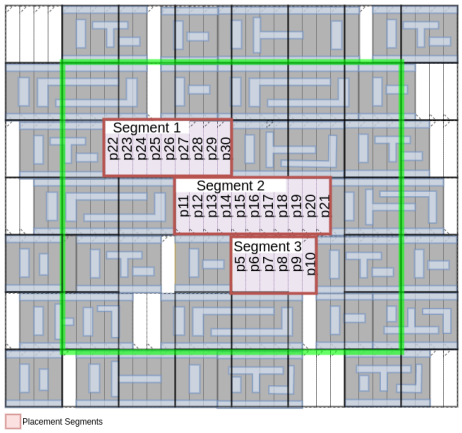
(b) Possible placement positions for movable cells inside the window

Cells	Possible Placement									·····			
	p1	p2	p3	p4	p5	p6	p7	p8	p9		p29	p30	p31
c1	✗	✗	✗	✗	✓	✓	✓	✗	✗	·····	✗	✗	✗
c2	✗	✗	✗	✗	✓	✓	✓	✗	✗		✗	✗	✗
c3	✗	✗	✗	✗	✓	✓	✓	✗	✗		✗	✗	✗
c4	✗	✗	✗	✗	✓	✓	✓	✗	✗		✗	✗	✗
c5	✗	✗	✗	✗	✓	✗	✗	✗	✗		✗	✗	✗

✗ Invalid Position ✓ Valid Position

(c) Exhaustive Search Table

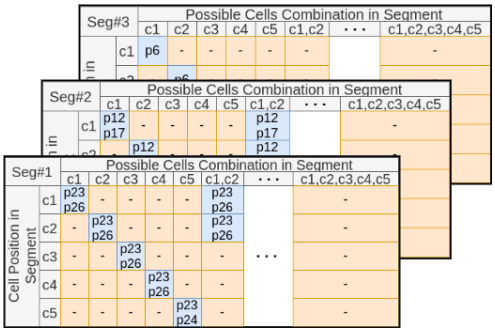
Figure 4.14: All possible placement positions for movable cells (yellow cells and a red cell in (a)) in a given detailed placement window (green box). (a) A window with its movable cells is shown. (b) All possible placement positions are highlighted (purple). (c) The table of all valid and invalid placements in an exhaustive search approach for each cell is given.



(a) Segmentation the available Space

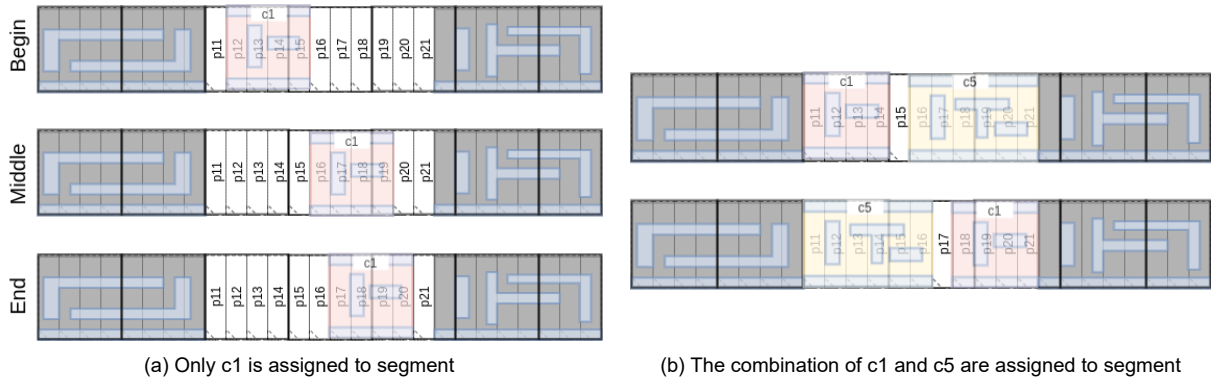
Segment	Possible Cell Combination in Segment							
	c1	c2	c3	c4	c5	c1,c2	·····	c1,c2,c3,c4,c5
Seg#1	✓	✓	✓	✓	✓	✓	·····	✗
Seg#2	✓	✓	✓	✓	✓	✓		✗
Seg#3	✓	✓	✓	✓	✓	✗		✗

(b) Possible Cell Combination in each Segmentation



(c) Possible Placement of cells in each Segment

Figure 4.15: ILP-DP Segmentation to reduce the size of ILP problem.



(a) Only c1 is assigned to segment

(b) The combination of c1 and c5 are assigned to segment

Figure 4.16: Placement assignment to each cell in a segment. To reduce the search space for assigning a position to a cell in a segment, a maximum of three positions are allowed. These three positions are selected according to the beginning, middle, and end of the segment. (a) shows a placement assignment to c1. (b) shows placement assignments to c1 and c5. When more than one cell is assigned to a cell all permutations of cells are investigated.

with few movable cells can be a huge search space. For instance, for 5 movable cells and 31 available sites, the search space for all permutations of cells is equal to 20,389,320. However, many of these available spots are invalid and can be removed from candidate placement for each cell. For instance, since the c1 width is 4, the placement p1 is not valid for c1. Therefore, the possible search space for each cell according to its width can be reduced. In Figure 4.14(c), the table of valid and invalid positions for the given window is presented. Therefore, to speedup the ILP model the input search space can be reduced if the possible positions are selected wisely. To address this issue, a pruning search space technique is proposed. In the pruning search technique, the following steps are taken:

1. The sequence of placement sites is partitioned into segments, as shown in Figure 4.15(a). A segment is defined as a part of a row that spans several empty sites. A list of empty sites is assigned to a segment if the number of sites in the segment is equal or more than the minimum width of the set of movable cells ($|sites| \geq min(W(movable_{set}))$). This guarantees that a segment is big enough to place a cell inside with no violation.

2. For each segment, the combination of cells that can fit in the segment is evaluated, shown in Figure 4.15(b).

3. For each combination of cells, all possible cell order permutations are calculated. The complexity of calculating all permutations for a given segment with k movable cells inside a segment with l available sites is $\frac{l!}{(l-k)!}$. The permutation will guarantee that placements with all orders of cells are evaluated.

4. Then, a position is assigned to each cell according to their order. This can be seen in Figure 4.15(c),

where for each cell and combination of cells inside the segment a position is assigned to each cell.

5. To keep the computations manageable, in cases where there are many empty spots available in a segment, a maximum of three positions is allowed for each cell in a segment. These three positions are selected according to the beginning, middle, and end of the segment, shown in Figure 4.16(a). Also, to reduce the possible conflict between abutting cells a white space is inserted between cells in the case that the white space is available on the segment. Figure 4.16 shows how the position assignment is done when more than one cell is assigned to a segment.

Using the above pruning search technique can reduce the number of candidate positions and speedup the "Generate Candidate Position" step in ILP-DP with a small impact on the quality of candidate positions.

4.4 Extension of Global Routing to Cooperate Routing with Cell Movement in CRP Engine

In this section, the routing used to estimate the quality of each placement and route of all nets are presented in detail. To improve the runtime of the global router and adapt global routing with cell movements, two caching techniques (called Cost catching technique and Net catching technique) are presented in Subsection 4.4.1 and Subsection 4.4.2, respectively. In CRP, the global router can be executed in two modes: Fixed Placement Mode (FPM) and Ripup Placement Mode (RPM). FPM is used to apply global routing in UD (Explained in Subsection 4.2.5), where the positions of the cells are fixed. RPM is used to estimate the cost of each candidate cell position in ECC (Explained in Subsection 4.2.3). The flow of FPM and RPM is presented in Figure 4.17.

In Figure 4.17, I highlight the main differences between the two modes of global routing. One key difference is that RPM includes a recording of initial routing and updating pin access, whereas FPM does not. After obtaining the pin access, the problem is converted into a 2D graph and routed using a 2D router (I use Flute [80] for this purpose). Based on the 2D routing solution, each net is classified into two groups: PatternRoute or AStar. If the edge of the 2D route intersects with any macro blockages in the layout, the net will be routed using AStar in 3D routing. Otherwise, it will be routed using PatternRoute. If the routed net has a violation in FPM mode, it will be pushed to a queue for ripup and reroute. In RPM mode, the cell position will be removed from candidate positions for the cell's movement. Although this approach can improve the global routing with cell movement, it is a runtime expensive task if it runs sequentially. In the following, to speedup the global routing algorithm in PatternRoute, a Cost Caching technique is proposed. Then, to parallel the flow of routing with cell movement a Net Caching technique is explained.

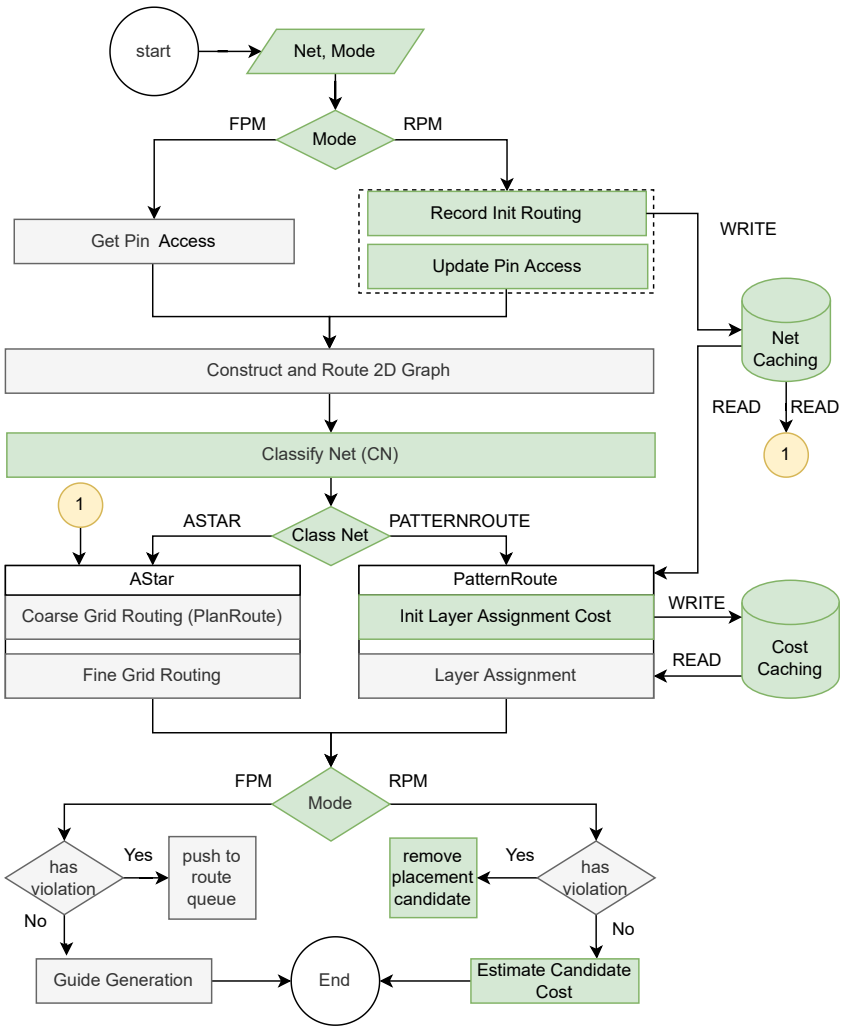


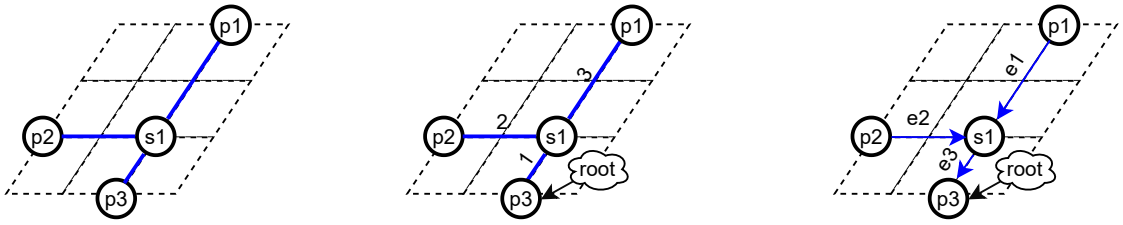
Figure 4.17: Flow of the 3D global routing used in CRP. The global routing can be executed in two modes: Fixed Placement Mode (FPM) and Rip-up Placement Mode (RPM). Note: Two routers inspired from CUGR [25] including AStar and PatternRoute are used.

4.4.1 Improve Runtime of Global Routing by Cost Caching Technique

The technique of Cost Caching involves storing costly function calls in a cache and recalling them if needed again. This can greatly improve the efficiency of the layer assignment problem in global routing, which is formulated as a dynamic programming problem, as demonstrated in the study by [116].

The problem of layer assignment for a single net is shown in Figure 4.18. To obtain a one-layer routing tree, the Flute algorithm [80] is used. A sample solution of the Flute algorithm for a net n with 3 pins ($Pin_n = \{p1, p2, p3\}$) and one Steiner point (s1) is shown in Figure 4.18 (a). In order to find the order of edges for the layer assignment problem, two steps are taken. First, an arbitrary node $p \in Pin_n$ as a root node is selected, and the Deep First Search (DFS) algorithm is used to traverse the routing tree. The edge order and a sample root node for net n are shown in Figure 4.18 (b). Second, for each edge, its edge order is

assigned in a reverse tree traversal and then it is transformed into directed edges, shown in Figure 4.18 (c). Three definitions are used to find the Minimum VIA Cost (MVC) for each vertex v: $child(v)$, $parent(v)$, and $parent_{edge}(v)$. $child(v)$ represents all children of vertex v, $parent(v)$ represents all parent nodes of vertex v, and $parent_{edge}(v)$ shows the parent edge of vertex v. In Figure 4.18 (c), as an example, $child(s1)$ are $p1$ and $p2$, $parent(s1)$ is $p3$, and $parent_{edge}(s1)$ is $e3$.



(a) One-layer Routing (Done by Flute) (b) Deep First Search (DFS) Traverse Tree (c) Edge Traversal Order

Figure 4.18: One-Layer Routing Edge Ordering

The $MVC(v, l)$ shows the minimum VIA cost of a sub-tree rooted at vertex v when $parent_{edge}(v)$ is assigned to layer l, and it can be calculated as follows:

$$MVC(v, l) = min((\sum_{v_j \in child(v)}^{|child(v)|} MVC(V_j, l_j)) + VCS(v, l_{via}, h_{via})) \tag{4.13}$$

The $\sum_{v_j \in child(v)}^{|child(v)|} MVC(V_j, l_j)$ represents the total VIA cost of all children of vertex v, where V_j is a child of vertex v. The $VCS(v, l_{via}, h_{via})$ shows the VIA Cost Stack (VCS) for node v, where l_{via} is the lowest (or bottom) VIA layer and h_{via} is the highest VIA layer assigned to the node v in the 3D graph. The stack of VIA cost is calculated as follows:

$$VCS(v, l_{via}, h_{via}) = \sum_{l \in l_{via}}^{h_{via}} VC(v, l) \tag{4.14}$$

The $VC(v, l)$ shows the VIA cost of vertex v in layer l, and it is calculated by Equation 4.5. For example, to calculate the $MVC(s1, 1)$, there are a total of $3 \times 3 = 9$ possible combinations. This is shown in Figure 4.19 (a). One possible solution is to connect nodes $p1$ and $p2$ to $s1$ in the first layer, shown in Figure 4.19 (b). In this example, since both children of node s1 (p1 and p2) used the first layer to connect to s1, therefore, the $VCS(p3, 1, 1)$ equals $VC(p3, 1)$. Another possible solution, shown in Figure 4.19 (c), is to connect nodes $p1$ and $p2$ from the first and third layers to node s1, respectively. In this figure, The $VCS(p3, 1, 3)$ equals the summation of VIA costs (or VIA cost stacks) $VC(p3, 1) + VC(p3, 2) + VC(p3, 3)$. This shows that for each possible combination to calculate $MVC(v, l)$, a different $VC(v, l)$ also can exist. Therefore, the

total complexity of finding $MVC(v,l)$, for a given l, requires all the combinations of $MVC(v_j,l_j)$'s for each $v_j \in child(v)$, and the l value is between 1 to L (the number of layers). Since the number of children of each vertex is a maximum of four, therefore, $MVC(v,l)$ can be done in $O(k^4)$ time. Thus, to compute the $MVC(v,l)$ for the given routing tree the total complexity is $O(k^4 \times k \times |V|) = O(k^5 \times |V|)$ time. This complexity shows the number of times that the minimum VIA cost for each node requires to be calculated. Calculating the VIA cost in layer assignment can be a runtime expensive task due to the probabilistic features of the proposed cost function (Equation 4.5).

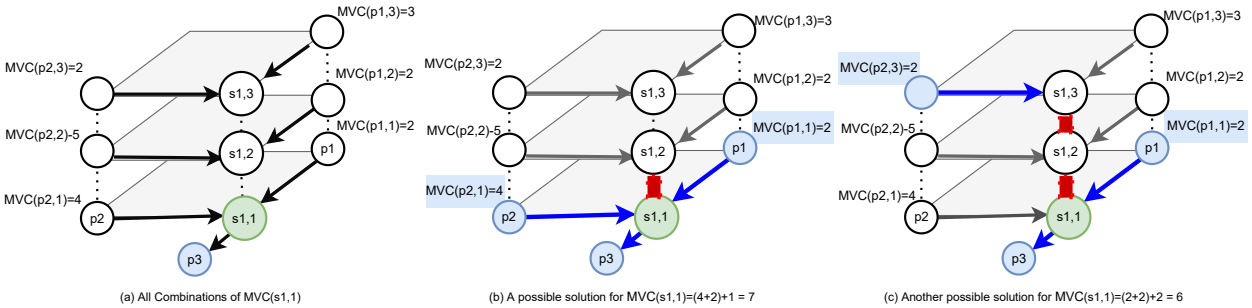
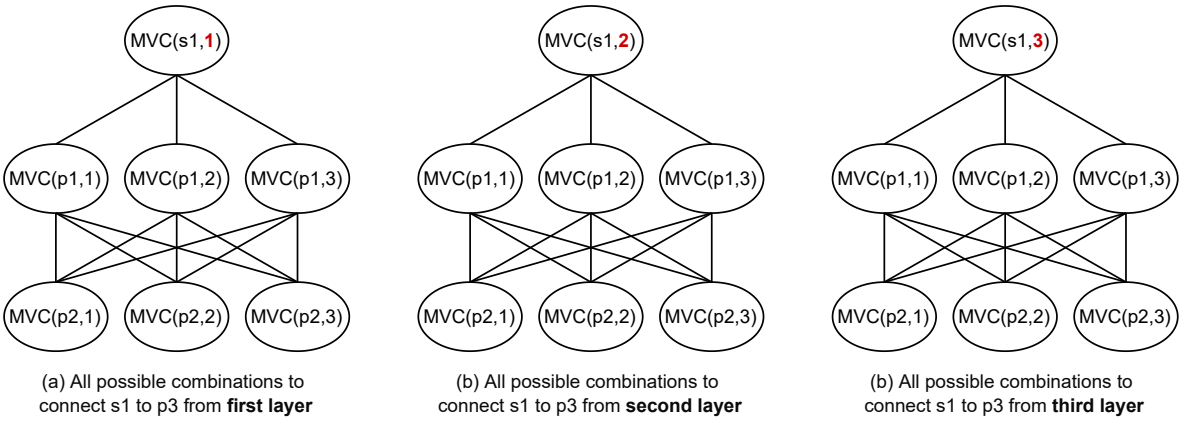


Figure 4.19: Dynamic Programming Layer Assignment Example. It is assumed the number of layers L is 3.



Cost Caching			
Address			Value
Node Index	Lowest Layer	Highest Layer	Via Cost (VC)
P1	1	1	VC(P1,1,1)
P1	1	2	VC(P1,1,2)
⋮	⋮	⋮	⋮
s1	3	3	VC(P1,3,3)

(d) Cost Caching Table

Figure 4.20: The computation tree of finding Minimum Via Cost (MVC) for Figure 4.19 (a) in (a)-(c). (d) Shows the Cost Caching Table.

To explain the frequency of calculating $MVC(v, l)$, the computation tree for the layer assignment problem of Figure 4.19 (a) is given in Figure 4.20. All combinations to connect s1 from different layers to p3 are shown in Figure 4.20 (a)-(c). The root node of the computation trees shows the minimum VIA cost of a sub-tree rooted at vertex s1 when $parent_{edge}(s1)$ is assigned to layers 1,2, or 3. Each level of the tree shows all the possible edges that can access different layers from a child node. For example, the node $MVC(p1, 1)$ shows that layer one is selected to connect p1 to s1. By finding the minimum path of these three trees, the result of the minimum cost of connecting s1 to p3 in a 3D grid graph can be calculated. However, as one can see, this computation tree can grow significantly if the number of nodes in the path and their children increases. Fortunately, this can be solved efficiently by dynamic programming [117] where the minimum cost of the current path depends on the minimum cost of its sub-path. Moreover, a more careful study shows the leaves of this computation tree are recalculating the same layer assignment VIA cost frequently. In total there are three layers (L=3) and two children for node s1 to connect p3. Therefore, there are $L \times L^{child(s1)} = 3 \times 3^2 = 27$ possible combination to find the MVC between s1 and p3. To avoid this recalculation of VIA cost, the results of VIA cost can be stored (or cached). The Cost Caching table has three columns as the address including Node Index, the lowest, and the highest layers assigned to the node, shown in Figure 4.20 (d). This Cost Caching table takes all the routing nodes of the 2D global routing (or one-layer routing) solution including the pin positions and Steiner points. Then, for each routing node point, all possible VIA Costs (VC) are calculated once and stored in a Cost Caching table. The first key is an index of each node in the constructed grid graph, and the second key is the bottom layer and highest layers assigned to the nodes of the routing tree. The Cost Caching table will be generated locally for each net and after routing the net will be released to save memory. This way, the global router can avoid recalculation of costs in the layer assignment step of the PatternRoute algorithm and improve the runtime.

4.4.2 Improve Runtime of Global Routing with Cell Movement by Net Caching Technique

A common approach in combining routing and placement is to sequentially remove the route of the nets connected to the cell, move the cell and then reroute all the connected nets to the cell [93] and [38]. However, this approach is hard to run in parallel and significantly time-consuming, especially for routing global nets. To solve this problem, I propose a Net Caching technique to keep track of routed nets. As one can see in Figure 4.21, Net Caching follows four main steps. Assume that a net N uses route 1 to connect C1 to C2. If C2 is moved from x to x' the following steps will be performed:

1. The routing information of net N (route 1) will be stored in a cache table indexed by layerIdx, GridIdx, EdgeIdx.

2. 3D global route in RPM mode will query from 3D usage map (shown in Figure 4.21).

3. The usage map gets the information of the previous routing topology and virtually removes the usage of route 1.

4. Finally, updated usage map information gets passed to the 3D pattern route and the router virtually generates route 2 for net N.

To develop Net Caching a hash table is used. This hash table has two benefits. First, if a movement is not a good candidate movement the initial routing solution of the net is retrieved easily instead of rerouting all the nets related to the cell. Second, the process can be run in parallel since a copy of the local area of the routing solution is generated and this will avoid updating the main data structure until the final step by keeping the track of each routing usage. Thus, the interaction with shared memory is minimized. Therefore, it helps to gain maximum performance from the global routing and placement engines.

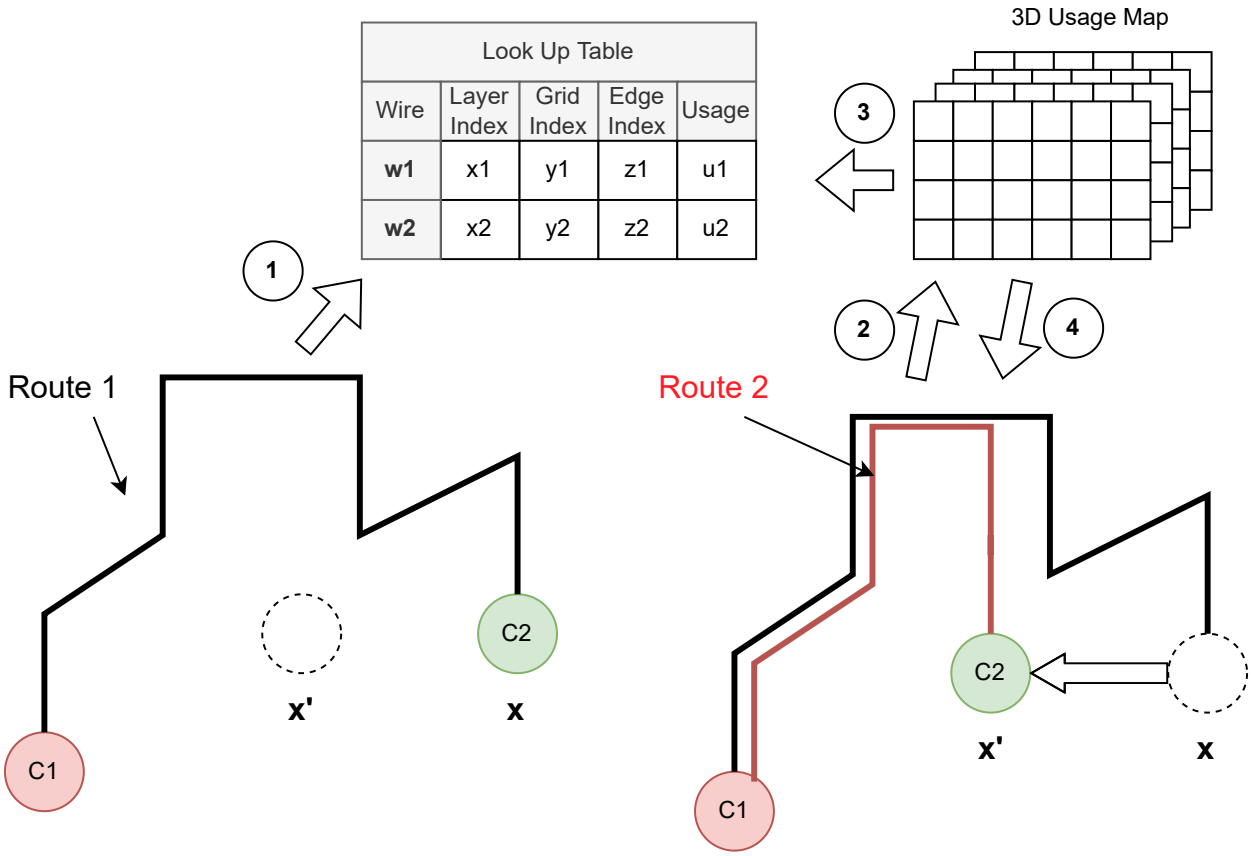


Figure 4.21: This Figure shows the Net Caching flow. By using net caching, routing with cell movement can be executed in parallel.

4.5 Summary

In this chapter, a framework for cooperation between routing and placement is proposed and the CRP engine is discussed. First, the proposed path cost model is designed to model the cost in routing and is explained in detail. Next, the five main steps of the CRP engine (LCC, GCP, ECC, FBC, and UD) are explained in detail. The proposed ILP-DP model for detailed placement and its flow, including the related cost model for each movement, are explained. Finally, the cost and net caching models, which improve global routing runtime and support parallel processing, are illustrated. In the next chapter, the effectiveness of the proposed framework is evaluated through experiments on ISPD 2018 and 2019 benchmarks.

Chapter 5

Experimental Results

To investigate the efficacy of the CRP engine and the proposed framework, a set of experiments was conducted and their results are presented in this chapter. In Section 5.1, the experimental setup is explained. In Section 5.2, the characteristics of two different benchmark sets used in the experiments, ISPD 2018 [42] and ISPD 2019 [43] contests, are discussed. The evaluation process and metrics are presented in Section 5.3, followed by experimental results for each benchmark in Section 5.4. Next, in Section 5.5, the techniques and methods used to improve the state-of-the-art are discussed. Finally, the performance of CRP consisting runtime and memory usage is reported in Section 5.6.

5.1 Experimental Setup

The CRP engine was designed and developed in C++. This engine benefits from several C++ packages such as Boost1.68 [102] and CPlex [110] to model geometry challenges and solve optimization problems. All experiments were done on a Linux desktop (Ubuntu 20.04) with an Intel Core i7-8700 CPU running at 3.20 GHz and 32 GB RAM with 8 threads.

5.2 Benchmark Characteristics

The proposed techniques were evaluated using the most recent benchmark suits available, ISPD 2018 and ISPD 2019. Each of these benchmark suits includes ten circuits described through Library Exchange Format (LEF) files, initial global routing guide files, and initial placement with the circuit's description (DEF) files. The LEF file includes the physical characteristics of the technology for cells (such as Standard cells, Macros, and IO Cells) and routing layers. The statistics of ISPD 2018 and ISPD 2019 are presented in Table 5.1. The

number of nets and cells for the circuits varies from 3k to 900k. Most circuits have 9 routing layers except ISPD 2019 test4 and test5 benchmarks which have 5 routing layers. Three technology nodes 65nm, 45nm, and 32nm are used. Each circuit contains different numbers and shapes of Macro blockages. A variety of the Placement Density (PD) from 9.48% to 100% [1] in ispd19_test5 to ispd18_test10 are given, respectively.

Table 5.1: ISPD 2018 and 2019 statistics. Columns 1 to 8 represent the name of each circuit, number of nets, number of cells, Placement Density (PD), technology node (Node) in nanometer, routing layers (L), number of macro blockages, and die size.

Benchmark	Nets (#)	Cells (#)	PD (%)	Node (nm)	L (#)	Macro (#)	Die Size (mm2)
ispd18_test1	9K	3K	85	45	9	0	0.20 x 0.19
ispd18_test2	36K	37K	57	45	9	0	0.65 x 0.57
ispd18_test3	36K	37K	65	45	9	4	0.99 x 0.70
ispd18_test4	72K	72K	89	32	9	4	0.89 x 0.61
ispd18_test5	72K	72K	92	32	9	8	0.93 x 0.92
ispd18_test6	108K	108K	99	32	9	0	0.86 x 0.53
ispd18_test7	180K	180K	90	32	9	16	1.36 x 1.33
ispd18_test8	192K	180K	90	32	9	16	1.36 x 1.33
ispd18_test9	193K	179K	91	32	9	0	0.91 x 0.78
ispd18_test10	290K	182K	100	32	9	0	0.91 x 0.87
ispd19_test1	9K	3K	83	32	9	0	0.14 x 0.14
ispd19_test2	72K	72K	72	32	9	4	0.87 x 0.58
ispd19_test3	8K	9K	84	32	9	4	0.19 x 0.19
ispd19_test4	146K	152K	21	65	5	7	1.60 x 1.55
ispd19_test5	29K	29K	9	65	5	6	0.90 x 0.90
ispd19_test6	180K	180K	75	32	9	16	1.35 x 1.32
ispd19_test7	360K	359K	96	32	9	16	1.58 x 1.51
ispd19_test8	540K	538K	79	32	9	16	1.80 x 1.70
ispd19_test9	899K	895K	84	32	9	16	2.00 x 2.15
ispd19_test10	899K	895K	88	32	9	16	2.00 x 2.15

The main differences between ISPD 2018 and ISPD 2019 are some added special nets in ISPD 2019 such as voltage nets (VDD) and ground nets (GND). This makes ISPD 2019 even more challenging than ISPD 2018, despite the higher placement density of the latter. On the other hand, unlike the ICCAD benchmarks, the ISPD benchmarks include detailed routing information and technology node specifications. The schematic of each benchmark is shown in Figure 5.1 and Figure 5.2, and further details are provided in Appendix C.

The ISPD benchmarks include advanced technology nodes challenges such as high cell utilization, complex pin access shapes, high routing congestion, boundary, and narrow channel routing issues, inner and boundary core blockages, and different technology nodes with a variety of physical design characteristics. For example, the pin access in the circuits can have complex shapes. For instance, in Figure 5.3, the ispd19_test5 has simple vertical rectangles for pin shapes (Figure 5.3 (A)), while the ispd18_test10 and ispd19_test3 have more complicated pin shapes with inner blockages (Figure 5.3 (B) and (C)). These features provide a wide range

[1] To achieve a 100% placement density in ISPD 2018 test10 benchmark, some blockages called FILLER are inserted in the layout. This is the main reason that Eh?Placer and ILP(Mov) failed in the experimental results due to the lack of infrastructure to model these components.

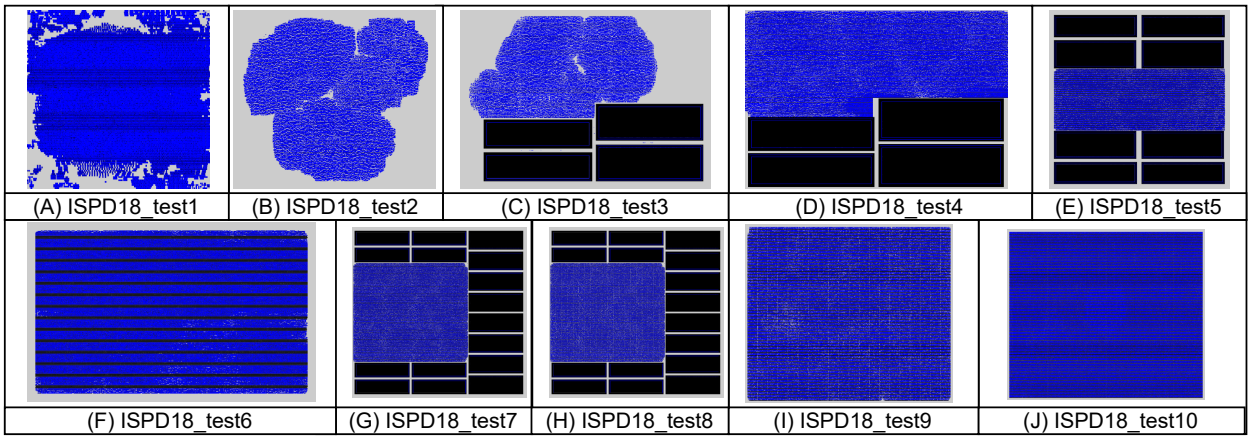


Figure 5.1: ISPD 2018 test1 to 10. The blue dots represent standard cells and the black rectangles shows the macro blockages.

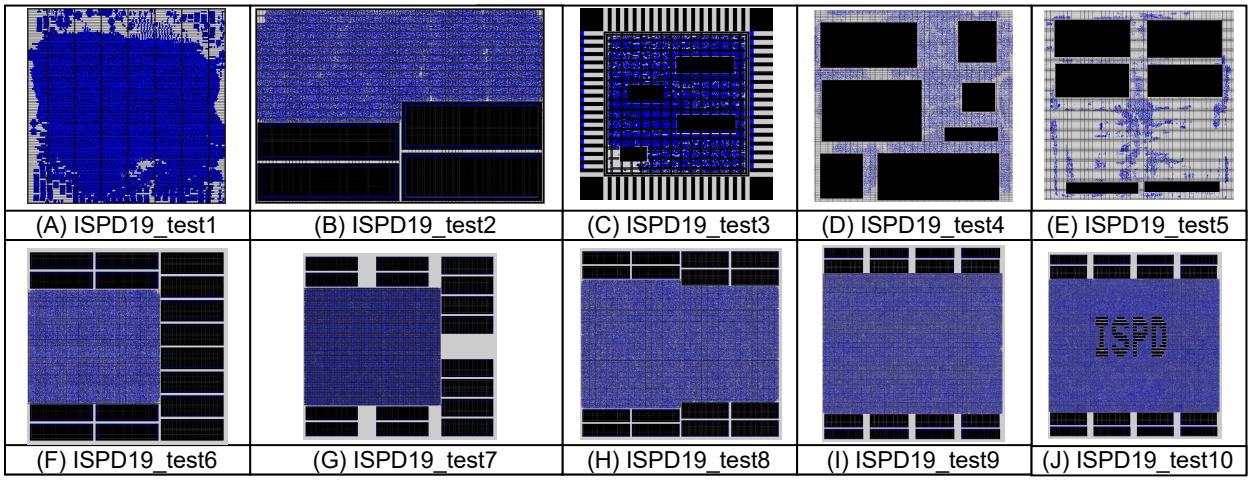


Figure 5.2: This Figure shows the ISPD 2018 test1 to test10 circuits. In this Figure, the blue rectangles represent standard cells and the black rectangles show the macro blockages.

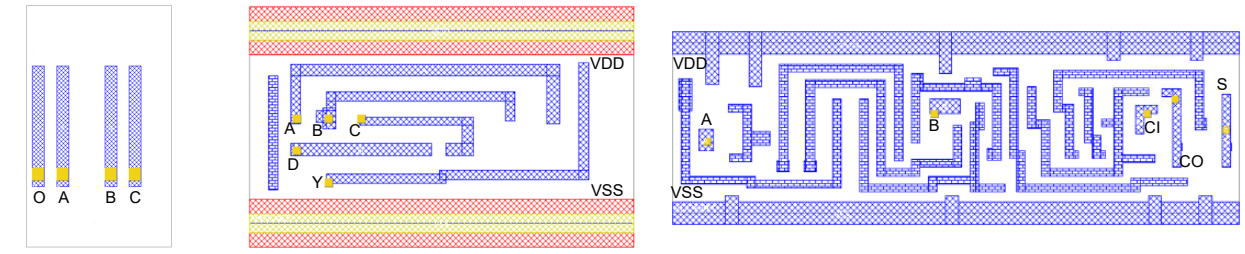


Figure 5.3: Pin access shapes complexity in different circuits. (A) Simple Pin Access shape with only vertical rectangles; (B) Complex pin access shape in NAND4x2; (C) Complex pin access with inner cell blockages in ADDFHX1.

of design styles that makes developing a framework that can improve all of the benchmark scores even more challenging. For further information about benchmarks please refer to [42] and [43].

5.3 Evaluation Process and Metrics

In this section, the evaluation metrics and evaluation process used on the ISPD benchmarks are discussed.

5.3.1 ISPD Evaluation Metrics

The CRP results were evaluated based on the detailed routing score reported by the official ISPD 2018 evaluator including connectivity constraints, routing metrics (physical wirelength [2] and VIA insertion), technology node rules (short, minimum area, and spacing rules), and routing preferences. Connectivity constraints must be satisfied to have a valid routing solution. This means that no open nets are allowed in a valid solution. One invalid solution is an open net which happens when a net is not fully connected to its pins. The short constraint happens if two routing objects including wire and VIA intersect each other. The minimum area and spacing define the minimum object area in each layer and the minimum spacing allowed from two metal objects in the same routing layer. Finally, the routing preferences include Out-of-Guide Wiring (OFGW), Out-of-Guide VIA (OFGV), Out-of-Track Wiring (OFTW), Out-of-Track VIA (OFTV), and Wrong Way Wiring (WWW). Note that the routing preference metrics are not hard rules meaning that reducing them can improve routing solution quality, but they do not need to be obeyed at all times. The evaluation metrics are given in Table 5.2. The raw score of the detailed routing can be measured by the summation of the metrics multiplied by their respective weights. All these constraints are defined by ISPD 2018 and 2019 contests to represent the technology node constraints and evaluate the detailed routing solution quality.

Table 5.2: The metric, abbreviation, and the weight of each ISPD metric.

Metric	Abbrevation	Weight	Metric	Abbrevation	Weight
Short Metal Area	Short	500	Total length of wires outside of the routing guides	OFGW	1
Number of Spacing Violations	Spacing	500	Total number of VIAs outside of the routing guides	OFGV	1
Number of min-area Violations	MinArea	500	Total length of off-track wires	OFTW	0.5
Total number of VIAs	VIAs	2	Total number of off-track VIAs	OFTV	1
Total length of wires	WL	0.5	Total length of wrong-way wiring	WWW	1

[2]Physical wirelength is related to the WL after detailed routing

5.3.2 ISPD Evaluation Process:

The evaluation process for the ISPD benchmarks is illustrated in Figure 5.4. For each benchmark, two input files of LEF and DEF are given. The DEF file includes the initial placement from the contest, and the LEF file includes technology node information. The flow can apply a detailed placement, a global routing, and a detailed routing to generate a new placement with respect to a detailed routing solution written in the DEF file, and a global routing solution written in the Guide file. Then, the DEF, Guide, and LEF files are sent to the official contest ISPD evaluator [42] to investigate the raw detailed routing score. Note: The weights in Table 5.2 are used in the evaluator to calculate the detailed routing raw score. Finally, a report file is generated by the evaluator that includes metric and final detailed routing scores. All experimental results in the next chapter are obtained from these report text files. For comparison, five algorithms are used to produce the input to the detailed routing solution. The different combinations to produce the final results are given in Figure 5.4 and are briefly discussed in the following.

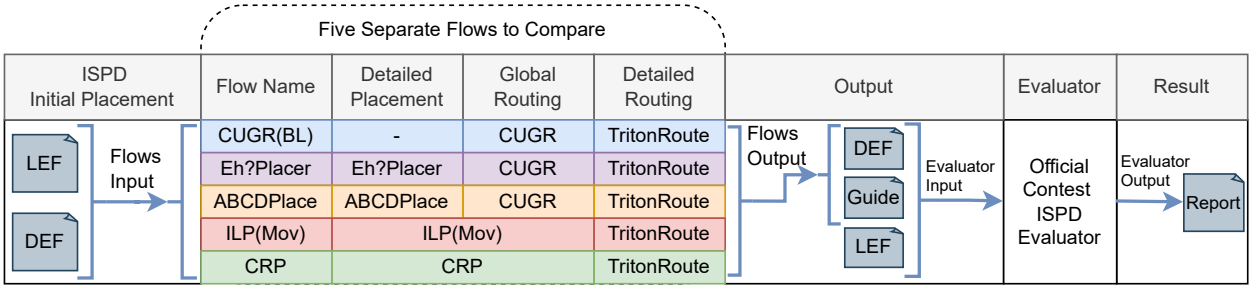


Figure 5.4: The evaluation process for ISPD benchmarks, and five separate flows to generate the comparison for the final routing solution results.

1. CUGR(BL): The initial placement from the ISPD is given to CUGR and TritonRoute without movement to generate the baseline for the comparisons. The BL in parentheses next to CUGR stands for 'Baseline'.

2. Eh?Placer: Eh?Placer [36] is used to improve the initial placement from the ISPD. CUGR and TritonRoute are used for global and detailed routing, respectively. Eh?Placer was chosen as it won second place in the ISPD 2014 contest [118] and the source code was available to me.

3. ABCDPlace: ABCDPlace [37] was used the improve the initial placement provided by the contest. ABCDPlace was chosen as it is a well-known academic placer, and the source code is obtained from [119].

4. ILP(Mov): ILP(Mov) [98] was used to generate global routing solutions from the initial ISPD Placement solutions (CUGR was not used). ILP(Mov) is applied since to the best of our knowledge, it is the only

work that applies routing with movement to the ISPD 2018 benchmarks. The ILP(Mov) binary to run on ISPD 2018 benchmark was available to us.

5. CRP: This is the proposed algorithm that applies movements to the initial ISPD placement and produces a global routing solution. TritonRoute is used to apply detailed routing in this flow.

Once the global routing results are produced, 20 iterations of Triton is used to produce the detailed routing solutions for all flows. Note: The size of the G-Cell considered for the global routing in all flows was 3000 DBU in all the benchmarks, except ISPD 2019 test4 and 5, the G-Cell size is 2000 DBU due to different technology nodes and row height.

5.4 Experimental Evaluation

In this section, the experimental results of the proposed framework are given for the ISPD 2018 and ISPD 2019 benchmarks. In Table 5.3 and Table 5.4, the breakdown of the results for the detailed routing metrics on ISPD 2018 and 2019 are reported. In both tables, the name of circuits (Benchmarks) and the name of flows (Flow) are shown in Columns 1 and 2. In Column 3, the number of movements is reported (#Moves). In Column 4, the routing metrics (wirelength (WL) and the number of VIAs (#VIAs)) are presented. The wirelength and #VIAs refer to actual wiring and via insertion after the detailed routing solution. In Columns 5 and 6, the routing preferences metrics and design rule violations (DRV) are given. Then, in Column 7 the raw score of the detailed routing solution is provided. The difference percentage from the best score in each circuit is shown in Column 8. In this column, the best score always has 0.00%. Finally, the rank of each flow is shown in Column 9 according to the detailed routing score. For example, the lowest detailed routing score among the flows in each circuit receives a rank of 1 and the highest detailed routing score has a rank of 4. All the results reported in Table 5.3 and Table 5.4 are obtained by the official evaluator of ISPD 2018 [42].

5.4.1 Experimental Resutls on ISPD 2018

In this section, the experimental results of CRP on ISPD 2018 benchmarks compared with CUGR(BL), Eh?Placer, ABCDPlace, and ILP(Mov) are given. The results in Table 5.3 show that CRP performs best to reduce WL, #VIAs, and OFGW in ISPD 2018 test1, test5,test6,test7,test8,test9, and test10, and corresponding to that CRP achieved the best detailed routing score in those benchmarks. The improvement of OFGW in these benchmarks shows a well-correlation between global routing and detailed routing. This means that wiring during the detailed routing step follows better the guides provided by global routing than the baseline flow (CUGR(BL)) in the benchmarks that OFGW is improved. Moreover, although, CRP

improved the detailed routing score compared with the baseline in ISPD 2018 test2 and test3, Eh?Placer performed best in these two benchmarks where the placement densities (PD) are less than 80%. The main reason CRP did not perform best in these two benchmarks is that CRP improves the routing issues locally by using local movement through iterations which results in a small improvement in the detailed routing in each iteration of CRP. This reveals the natural trade-off between net modeling and placement density. In other words, in benchmarks with low placement density gentle net modeling like HPWL can be good net modeling while in benchmarks with high placement density higher correlated net modeling like global routing is required to be used. Moreover, there can be a miscorrelation between global routing and detailed routing. This shows in ispd18_test4, where CUGR(BL) performed best compared with CRP and other flows. This represents a miscorrelation between global routing and detailed routing in this benchmark and using global routing feedback for each placement may not guarantee detailed routing score improvement. The main reason that there is a mismatch between global routing and detailed routing is the core and die area are not well aligned. Finally, These results show that the proposed methods and techniques developed in CRP could achieve better results in the bigger circuits with higher placement density and routing congestion. This improvement in high-density circuits was obtained thanks to the proposed CM-TCM (defined in Definition 4.7) and the well-correlation between global routing with detailed routing.

5.4.2 Experimental Results on ISPD 2019

In this section, the experimental results of CRP on ISPD 2019 benchmarks compared with CUGR(BL), Eh?Placer, and ABCDPlace are given. At the time these results are produced, the ILP(Mov) framework was not compatible with ISPD 2019 benchmarks. The results reported in Table 5.4 show that CRP is able to perform best in detailed routing scores in ISPD 2019 benchmarks compared to the others except for test2 and test6, where Eh?Placer has a smaller detailed routing score [3]. Moreover, CRP could generate less VIA insertion and wirelength in most of the benchmarks without sacrificing design rule violations. This shows that the proposed methods in CRP are able to improve the detailed routing scores and advanced technology nodes challenges compared with the best-known existing frameworks. To investigate further how this improvement was achieved, a comparison of CRP with each flow is discussed in detail in Section 5.5. Note: CRP failed in ispd19_test4 during the detailed routing. This presents one of the weaknesses of the CRP engine which is a strong dependence on global routing solution quality. If the correlation between global and detailed routing is poor, the movements in CRP can result in an unroutable solution.

[3]These two benchmarks have less than 80% placement densities.

Table 5.3: The breakdown of results for ISPD 2018 for the detailed routing solution in five flows is reported. Column 1 to 16 shows the name of a circuit, the flow name, the number of movements (# Moves), the routing metrics (such as wirelength and VIAs), the routing preference metrics, Design Rule Violations (DRVs), raw detailed routing score, the difference percentage from the first score in each circuit, and the raw detailed routing score rank.

Benchmark	Flow	# Moves	Routing Metrics		Routing Preference Metrics					DRV			Score	Score Diff (%)	Rank
			WL	# VIAs	OFGW	OFGV	OFTW	OFTV	WWW	Short	MinArea	Spacing			
ispd18_test1	CUGR(BL)	0	171M	38K	2M	0K	1M	0K	3M	0	0	0	302K	0.88%	2
	Eh?Placer	6173	176M	38K	2M	0K	1M	0K	3M	0	0	0	310K	3.45%	3
	ABCDPlace	4972	176M	38K	2M	0K	1M	0K	3M	0	0	0	310K	3.48%	4
	ILP(Mov)	622	181M	37K	3M	0K	0M	0K	3M	0	0	0	317K	5.87%	5
	CRP2.0	**28**	**170M**	**37K**	**1M**	0K	1M	0K	3M	0	0	0	**299K**	**0.00%**	**1**
ispd18_test2	CUGR(BL)	0	3124M	381K	21M	3K	6M	2K	30M	0	0	0	4,810K	1.39%	4
	Eh?Placer	19613	3068M	376K	26M	4K	6M	2K	31M	0	1	0	4,744K	0.00%	1
	ABCDPlace	7183	3153M	383K	26M	4K	7M	2K	31M	0	0	0	4,863K	2.51%	5
	ILP(Mov)	359	3135M	359K	30M	5K	4M	2K	32M	0	0	0	4,803K	1.26%	3
	CRP2.0	492	3129M	371K	20M	4K	7M	2K	33M	0	0	0	4,799K	1.17%	2
ispd18_test3	CUGR(BL)	0	3512M	379K	22M	3K	6M	2K	29M	0	0	0	5,290K	1.15%	4
	Eh?Placer	20004	3461M	372K	27M	4K	6M	2K	31M	0	0	0	5,230K	0.00%	1
	ABCDPlace	5397	3531M	378K	26M	4K	7M	2K	31M	0	0	0	5,325K	1.82%	5
	ILP(Mov)	468	3506M	357K	32M	5K	5M	2K	32M	0	0	0	5,270K	0.77%	2
	CRP2.0	525	3510M	368K	22M	4K	7M	2K	33M	0	0	0	5,278K	0.92%	3
ispd18_test4	CUGR(BL)	0	5240M	719K	27M	8K	10M	17K	28M	0	0	0	14,863K	0.00%	1
	Eh?Placer	42440	5317M	718K	35M	9K	9M	17K	28M	0	0	0	15,094K	1.55%	5
	ABCDPlace	10093	5312M	725K	35M	9K	9M	18K	28M	0	0	0	15,091K	1.53%	4
	ILP(Mov)	1009	5243M	720K	27M	8K	10M	17K	28M	0	0	0	14,872K	0.05%	2
	CRP2.0	346	5245M	716K	31M	9K	9M	17K	29M	0	0	0	14,895K	0.22%	3
ispd18_test5	CUGR(BL)	0	5485M	844K	21M	9K	4M	15K	17M	0	0	0	15,624K	0.40%	3
	Eh?Placer	41006	5623M	835K	25M	8K	3M	15K	18M	0	0	0	15,973K	2.64%	5
	ABCDPlace	9354	5588M	832K	21M	7K	3M	15K	18M	0	0	0	15,861K	1.92%	4
	ILP(Mov)	1223	5485M	843K	21M	8K	4M	15K	17M	0	0	0	15,618K	0.36%	2
	CRP2.0	**331**	**5483M**	**818K**	**18M**	7K	3M	15K	19M	0	0	0	**15,562K**	**0.00%**	**1**
ispd18_test6	CUGR(BL)	0	7100M	1,278K	31M	10K	4M	23K	24M	0	0	0	20,627K	0.44%	2
	Eh?Placer	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	5
	ABCDPlace	12950	7761M	1,284K	32M	12K	6M	23K	26M	0	0	0	22,310K	8.64%	4
	ILP(Mov)	1943	7116M	1,283K	31M	10K	4M	23K	24M	0	0	0	20,676K	0.68%	3
	CRP2.0	**130**	**7090M**	**1,246K**	**29M**	11K	6M	23K	25M	0	0	0	**20,536K**	**0.00%**	**1**
ispd18_test7	CUGR(BL)	0	12918M	2,096K	49M	16K	7M	28K	37M	0	0	0	36,980K	0.34%	2
	Eh?Placer	93687	13111M	2,070K	55M	19K	10M	29K	39M	0	0	0	37,459K	1.63%	5
	ABCDPlace	19787	13086M	2,066K	51M	18K	10M	29K	40M	0	0	0	37,373K	1.40%	4
	ILP(Mov)	2698	12928M	2,097K	49M	16K	7M	28K	38M	0	0	0	37,009K	0.41%	3
	CRP2.0	**881**	**12909M**	**2,039K**	**45M**	18K	10M	29K	42M	0	0	0	**36,856K**	**0.00%**	**1**
ispd18_test8	CUGR(BL)	0	13033M	2,147K	58M	18K	8M	29K	39M	0	0	0	37,424K	0.69%	2
	Eh?Placer	89307	13175M	2,108K	62M	21K	11M	29K	40M	0	0	0	37,739K	1.54%	5
	ABCDPlace	24960	13147M	2,108K	59M	20K	10M	29K	40M	0	0	0	37,655K	1.31%	4
	ILP(Mov)	2688	13044M	2,146K	56M	18K	8M	29K	39M	0	0	0	37,445K	0.75%	3
	CRP2.0	**1018**	**12989M**	**2,077K**	**51M**	19K	10M	29K	42M	0	0	0	**37,168K**	**0.00%**	**1**
ispd18_test9	CUGR(BL)	0	10808M	2,150K	58M	18K	7M	28K	38M	0	0	0	31,863K	0.76%	2
	Eh?Placer	101164	10900M	2,100K	58M	19K	10M	28K	38M	0	0	0	32,004K	1.21%	5
	ABCDPlace	27008	10891M	2,098K	54M	18K	10M	28K	38M	0	0	0	31,956K	1.05%	4
	ILP(Mov)	2894	10815M	2,150K	57M	18K	7M	28K	38M	0	0	0	31,874K	0.79%	3
	CRP2.0	**1119**	**10793M**	**2,065K**	**47M**	19K	10M	29K	41M	0	0	0	**31,623K**	**0.00%**	**1**
ispd18_test10	CUGR(BL)	0	13559M	2,310K	154M	54K	13M	32K	52M	0	0	0	39,666K	0.53%	2
	Eh?Placer	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	5
	ABCDPlace	14519	16169M	2,300K	210M	68K	18M	33K	56M	19,044k	500	0	46,984K	19.07%	3
	ILP(Mov)	2613	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	4
	CRP2.0	**261**	**13531M**	**2,245K**	**150M**	53K	16M	33K	53M	24k	0	0	**39,459K**	**0.00%**	**1**

Table 5.4: The breakdown of results for ISPD 2019 for the detailed routing solution in five flows is reported. Column 1 to 16 shows the name of a circuit, the flow name, the number of movements (# Moves), the routing metrics (such as wirelength and VIAs), the routing preference metrics, Design Rule Violations (DRVs), raw detailed routing score, the difference percentage from the first score in each circuit, and the raw detailed routing score rank.

Benchmarks	Flow	# Moves	Routing Metrics		Routing Preference Metrics					DRV			Score	Score Diff (%)	Rank
			WL	# VIAs	OFGW	OFGV	OFTW	OFTV	WWW	Short Area	# MinArea	Spacing			
ispd19_test1	CUGR(BL)	0	126M	41K	1M	1K	0.3M	4.0K	2M	0	0	0	416k	0.21%	2
	Eh?Placer	6173	157M	41K	3M	1K	0.5M	3.9K	2M	0	0	0	503k	21.22%	4
	ABCDPlace	3729	132M	39K	2M	0K	0.4M	4.0K	2M	0	0	0	433k	4.37%	3
	CRP2.0	**31**	**126M**	**38K**	2M	1K	0.4M	4.0K	2M	0	0	0	**415k**	**0.00%**	**1**
ispd19_test2	CUGR(BL)	0	4,967M	804K	79M	26K	16.3M	132.5K	33M	0	3	0	14,787k	2.58%	3
	Eh?Placer	39372	4,838M	795K	72M	20K	15.0M	134.6K	35M	0	0	8	14,415k	0.00%	1
	ABCDPlace	13698	4,998M	803K	74M	22K	14.4M	133.2K	34M	0	0	0	14,833k	2.90%	4
	CRP2.0	1766	4,940M	792K	64M	19K	14.3M	132.3K	33M	0	0	0	14,609k	1.34%	2
ispd19_test3	CUGR(BL)	0	165M	64K	6M	2K	0.6M	6.1K	3M	0	0	0	594,807k	0.77%	2
	Eh?Placer	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	4
	ABCDPlace	2154	171M	64K	7M	2K	0.5M	6.0K	3M	7308K	0	14	712k	20.73%	3
	CRP2.0	**303**	**164M**	**63K**	6M	2K	0.5M	6.1K	3M	0	0	0	**590k**	**0.00%**	**1**
ispd19_test4	CUGR(BL)	0	5,990M	1,074K	386M	168K	14.4M	1.8K	86M	487,845K	0	18,943	35258k	0.00%	1
	Eh?Placer	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	4
	ABCDPlace	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	3
	CRP2.0	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	2
ispd19_test5	CUGR(BL)	0	966M	166K	28M	15K	2.0M	0.2K	11M	322,570K	0	14	7,001k	147.63%	2
	Eh?Placer	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	4
	ABCDPlace	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	3
	CRP2.0	**2568**	**965M**	**156K**	10M	4K	1.1M	0.1K	9M	**0**	0	0	**2,827k**	**0.00%**	**1**
ispd19_test6	CUGR(BL)	0	13,146M	2,130K	75M	26K	14.0M	46.7K	53M	0	0	0	37,874k	4.94%	4
	Eh?Placer	95337	12,477M	1,963K	109M	29K	12.1M	48.3K	64M	0	0	5	36,090k	0.00%	1
	ABCDPlace	39574	13,144M	1,965K	103M	27K	12.0M	47.5K	61M	0	0	2	37,715k	4.50%	3
	CRP2.0	**4760**	**13,104M**	**1,956K**	91M	25K	12.1M	46.2K	59M	0	3	2	37,526k	3.98%	**2**
ispd19_test7	CUGR(BL)	0	24,286M	3,813K	307M	111K	35.0M	255.4K	172M	0	0	0	71,187k	0.96%	2
	Eh?Placer	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	4
	ABCDPlace	43170	24,369M	3,809K	289M	94K	34.6M	255.7K	174M	0	0	0	71,291k	1.11%	3
	CRP2.0	**1920**	**24,173M**	**3,782K**	249M	81K	34.2M	255.0K	170M	0	0	0	**70,512k**	**0.00%**	**1**
ispd19_test8	CUGR(BL)	0	37,132M	6,658K	197M	70K	15.6M	385.0K	156M	0	0	0	108,406k	0.68%	2
	Eh?Placer	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	4
	ABCDPlace	70149	37,404M	6,195K	251M	89K	45.3M	389.2K	191M	0	0	1	108,700k	0.96%	3
	CRP2.0	**803**	**37,111M**	**6,173K**	221M	79K	44.1M	382.3K	173M	0	0	5	**107,670k**	**0.00%**	**1**
ispd19_test9	CUGR(BL)	0	56,094M	11,070K	333M	120K	30.7M	641.9K	267M	0	0	0	166,215k	0.79%	2
	Eh?Placer	296783	57,928M	10,324K	445M	154K	82.0M	661.7K	326M	0	0	4	170,350k	3.29%	4
	ABCDPlace	89934	56,614M	10,305K	413M	149K	78.5M	648.5K	318M	0	0	1	166,792k	1.14%	3
	CRP2.0	**413**	**56,035M**	**10,250K**	376M	135K	77.0M	637.6K	296M	0	0	8	**164,918k**	**0.00%**	**1**
ispd19_test10	CUGR(BL)	0	55,609M	9,599K	558M	180K	86.0M	637.5K	427M	6,484K	0	91	164,305k	0.49%	2
	Eh?Placer	188875	57,859M	9,712K	622M	181K	89.0M	650.3K	450M	1K	0	25	170,497k	4.28%	4
	ABCDPlace	71952	55,990M	9,639K	600M	177K	85.3M	644.0K	441M	0	0	43	165,512k	1.23%	3
	CRP2.0	**397**	**55,448M**	**9,568K**	518M	150K	84.0M	638.0K	424M	0	0	73	**163,498k**	**0.00%**	**1**

5.5 Discussion

5.5.1 CRP vs Eh?Placer and ABCDPlace (Improvement of 3D Global Routing vs HPWL):

In this section, the difference between Eh?Placer and ABCDPlace (the state-of-the-art placements) compared with CRP are discussed. One of the main differences between these frameworks is the definition of the objective function after each cell's movement. Since there can be thousands to millions of cells in circuits with different placement densities, a poor model of this objective function during placement can cause a significant miscorrelation between the placement solution and the detailed routing solution. As an example, this miscorrelation is shown in Figure 5.5. In the first figure (Figure 5.5(a)), the percentage differences between HPWL in Eh?Placer and CRP compared with the baseline (CUGR(BL)) is presented. As can be seen in this figure, HPWL is improved after placement performed by Eh?Placer, the main reason is that in Eh?Placer (like ABCDPlace) cells are moved to reduce HPWL of the nets. In other words, the nets' wirelength is modeled as HPWL. However, the wirelength and #VIAs improvement compared with the baseline get worst when HPWL is reduced in most of the benchmarks, as shown in Figure 5.5(b) and (c), except in ispd18_test2 and test3, where the Placement Density is less than 80% (shown in Column 4 in Table 5.1).

In contrast, the HPWL changes produced by CRP are minor in most of the benchmarks and even it can be seen in Figure 5.5(a), it had an increase in HPWL to improve VIA insertion and keep minor changes in the wirelength. These improvements in wirelength and VIAs are shown in Figure 5.5(b) and (c). The main reason is that in CRP cells are moved to reduce the 3D global routing path cost (the path cost is a function of VIA, wirelength, and congestion described in Section 4.1). This shows that the objective function used in CRP has a better correlation with the detailed routing compared with HPWL, especially in benchmarks with high placement densities. This also shows that reducing HPWL in benchmarks with high placement densities does not necessarily cause improvement in VIA insertion and wiring in the detailed routing solution. Therefore, considering HPWL to evaluate the performance of a placement algorithm in modern technology nodes can not be considered a well-correlated metric with the detailed routing solution.

5.5.2 CRP vs ILP(Mov)

Although ILP-based routing with cell movement (ILP(Mov)) and CRP follow the same idea of routing with movement, there are two main differences between these two algorithms. First, CRP always returns a legalized solution after each movement, while in ILP(Mov) cells may overlap with each other following a

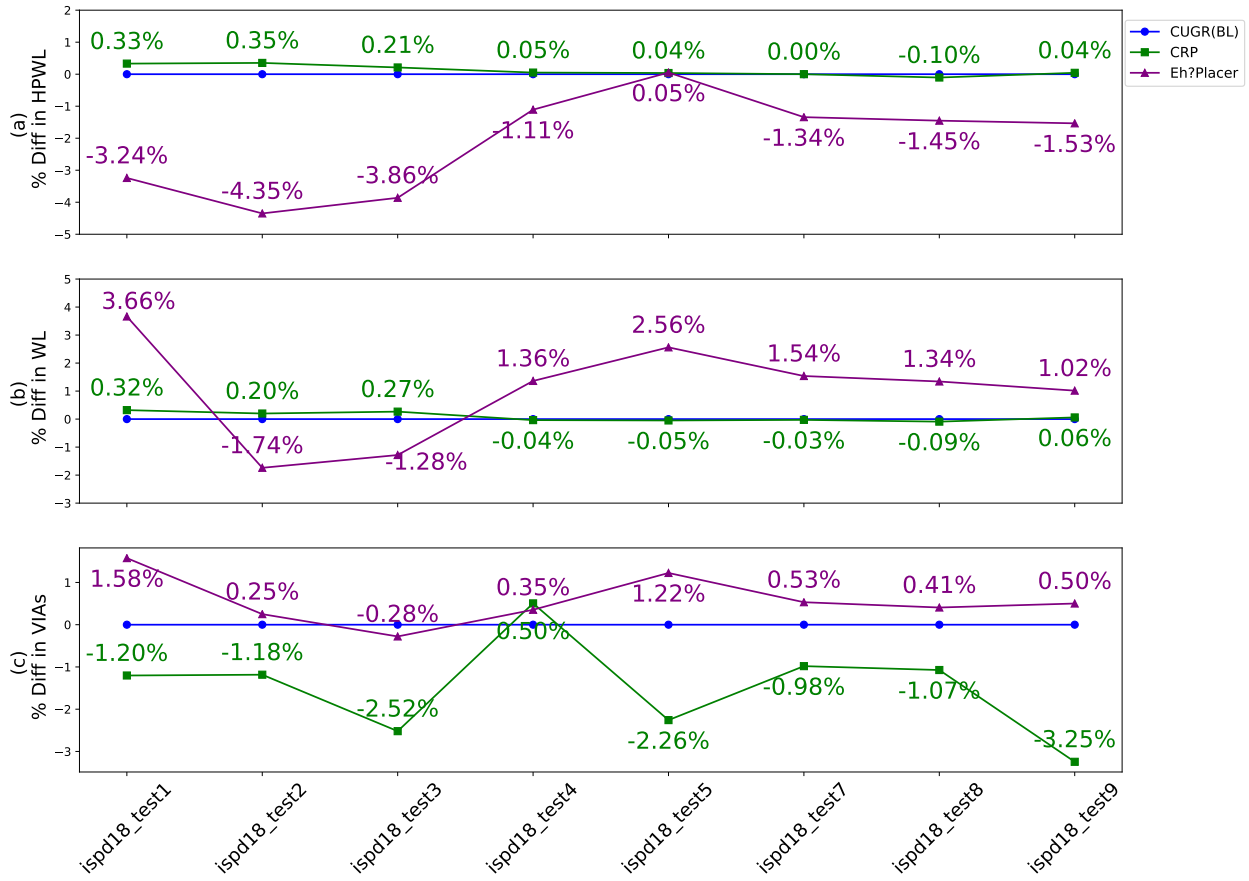
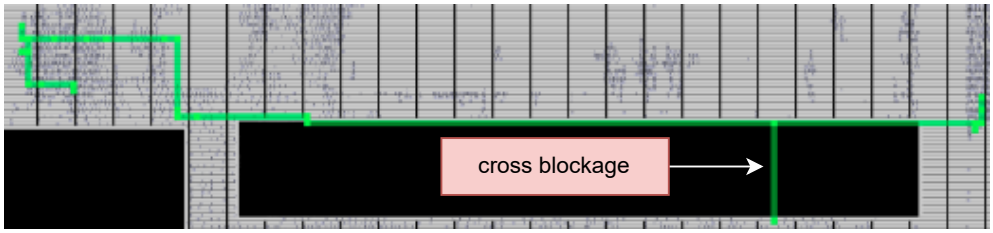


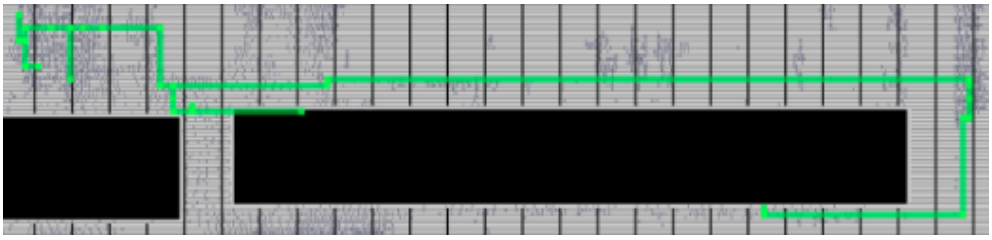
Figure 5.5: The percentage difference of HPWL(a), physical wirelength (WL)(b), physical VIA insertion(c) from the baseline with no movement. Note: Negative percentage means improvement compared with the baseline. Each color shows the flow in Figure 5.4.

movement, and thus, those overlaps are fixed by a commercial legalizer in a post-processing step. Second, in CRP, two routers (such as PatternRoute and AStar) are used due to layout complexity, while ILP(Mov) only uses PatternRoute routing. The importance of using different routers is presented in Figure 5.6. It can be seen in this figure, when cells are distributed around macro blockages, using PatternRoute (Figure 5.6(a)) can not guarantee a valid global routing solution. In addition, the use of PatternRoute may cause an increase in OFGW and runtime overhead in the detailed routing. Routing these types of nets utilizing AStar (Figure 5.6(b)) is required so as to provide a valid global routing solution. Therefore, due to the different characteristics of nets, in advanced technology nodes using different routing algorithms is essential. The impact of net classification on CRP is demonstrated by comparing CRPv1 and CRPv2. CRPv1 only employs the PatternRoute method to estimate the cost of each movement, while CRPv2 benefits from both PatternRoute and AStar methods through the use of a classification technique for cost estimation. According to this table, CRPv2, which utilized the proposed net classification technique, achieved better

detailed routing scores in the majority of the benchmarks..



(a) PatternRoute



(b) Astar Routing

Figure 5.6: An example of routing of the same net with PatternRoute (a) and AStar (b). The routed net is the net **n_4450** from ispd19_test5 circuit.

Table 5.5: The impact of net classification on CRP. **CRPv1:** only employs the PatternRoute method to estimate the cost of each movement. **CRPv2:** benefits from both PatternRoute and AStar methods through the use of a classification technique for cost estimation.

benchmarks	DR Score (DBU)		benchmarks	DR Score (DBU)	
	CRPv1 (10^3)	CRPv2 (10^3)		CRPv1 (10^3)	CRPv2 (10^3)
ispd18_test1	301	**299**	ispd18_test6	20606	**20536**
ispd18_test2	4800	**4799**	ispd18_test7	36952	**36856**
ispd18_test3	5283	**5278**	ispd18_test8	37380	**37168**
ispd18_test4	**14862**	14895	ispd18_test9	31818	**31623**
ispd18_test5	15623	**15562**	ispd18_test10	39579	**39459**

5.5.3 CRP vs Baseline (CUGR(BL))

The main purpose of using CRP in detailed routing is to improve the routing score by minimizing the number of VIA insertions by relocating cells to new routing spaces in the lower layers (first three metal layers). This is demonstrated in Figure 5.7, which compares the wirelength in CRP with the baseline. The figure shows that the wire usage in layers 1 and 2 is 158 million DBU and 105 million DBU more than the baseline, respectively. However, the middle layers 3, 4, and 5 have less wire usage in CRP.

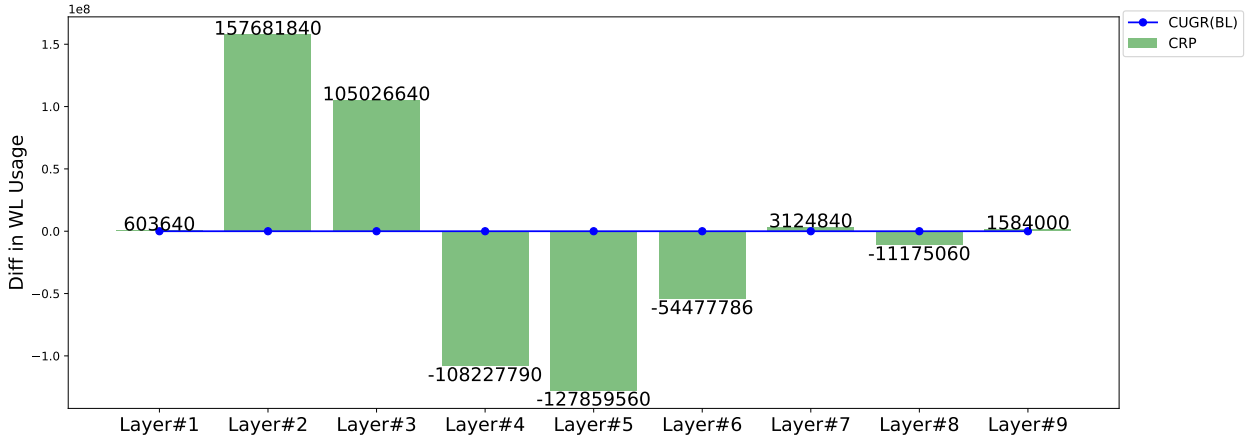


Figure 5.7: The difference in wire usage in each layer in ispd18_test7. CRP encourages routing to the lower layers by movement.

5.5.4 Path Cost vs Wirelength Absolute Cost

Figure 5.8 illustrates the significance of selecting an appropriate cost function when evaluating the impact of cell movement. It compares the improvement achieved by CRP when using two different cost functions: the absolute wirelength (WL ABS Cost), which aims solely at minimizing wirelength, and the path cost, proposed in Equation 4.8. Considering wirelength as the main objective can reduce wirelength compared with the path cost, as shown in Figure 5.8(a). However, the second graph shows that path cost performs better than wirelength absolute cost for VIA improvement (Figure 5.8(b)). The main difference between the improvement of wirelength absolute cost versus path cost is shown in OFGW and short violation, Figure 5.8(c) and (d) respectively. Considering only the improvement of the wirelength results in a huge increase in the detailed routing raw score. This means that considering only a single objective like wirelength can degrade the quality of other metrics. It is important to mention that routing and placement in the physical design by nature are multi-objective problems and considering an algorithm that only can improve one single objective can be an inaccurate approach. This also shows the importance of the proposed path cost which is a function of wirelength, VIAs, and congestion (described in Section 4.1) in evaluating each movement.

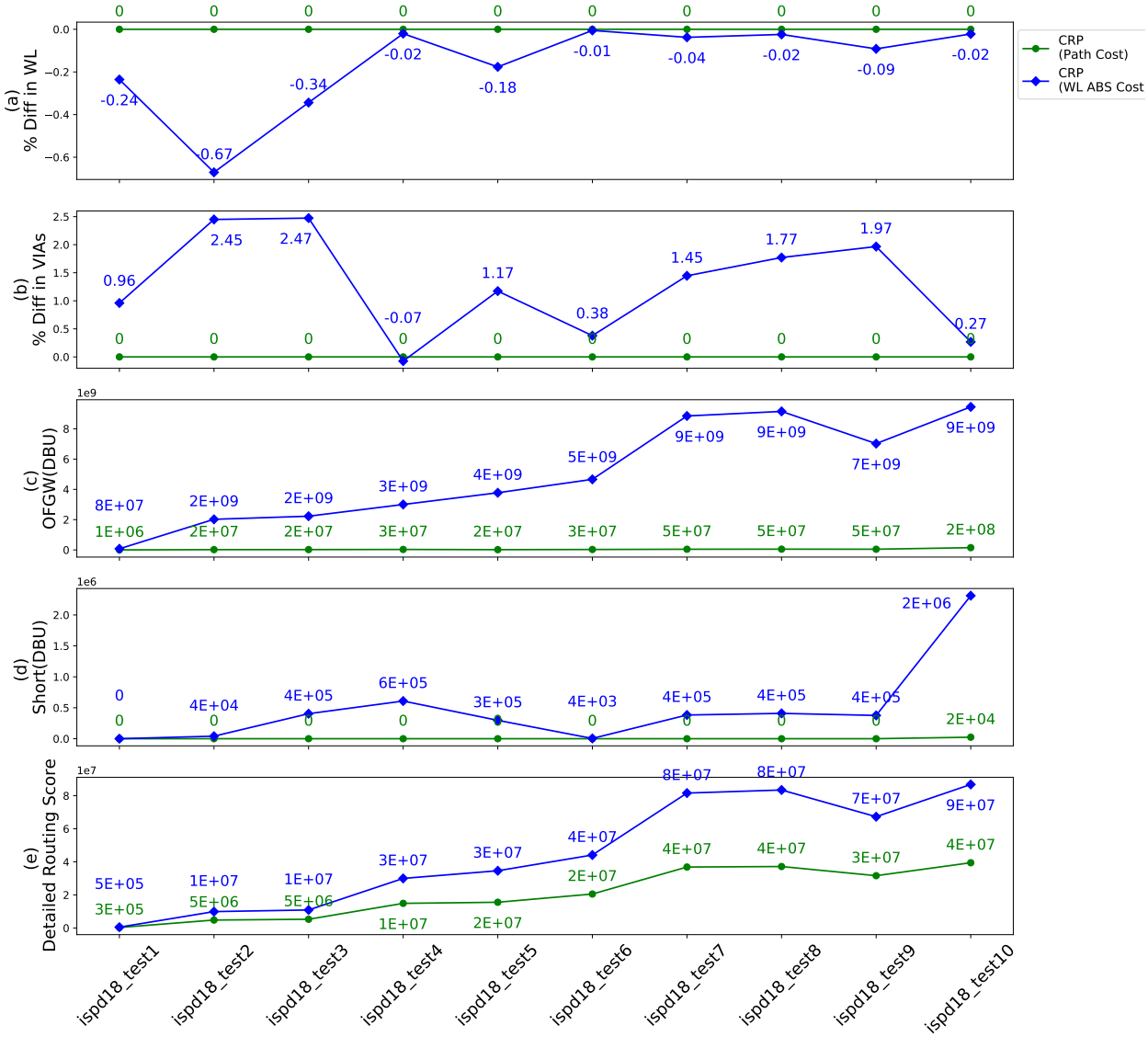
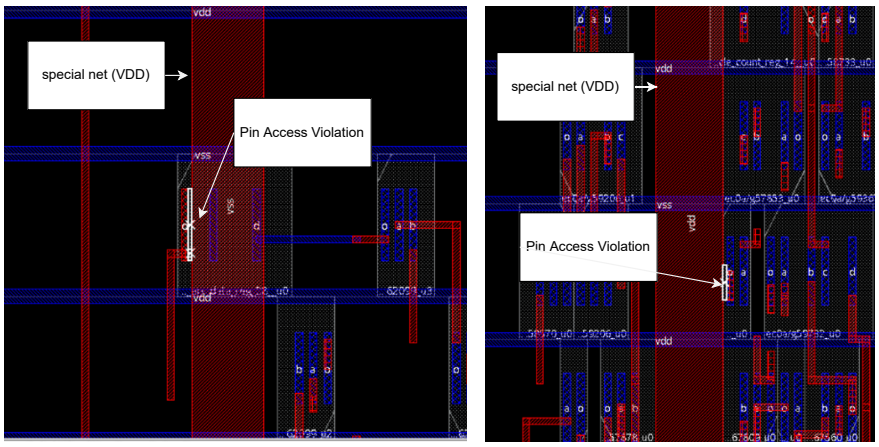


Figure 5.8: Illustration of the performance of CRP with different cost functions. CRP with reducing path cost function compared with CRP reducing wirelength absolute value after each movement. Figures (a) and (b) show the percentage difference between wirelength and VIAs of CRP(WL ABS Cost) from CRP(Path Cost). The negative percentage shows an improvement in the metrics. The actual value of OFGW, Short, and Detailed Routing Score in these two scenarios are shown in Figures (c), (d), and (e).

5.5.5 Routing with Cell Movement in The Presence of Special Nets

One of the main differences between the ISPD 2018 and ISPD 2019 benchmarks is the addition of special nets (VDD/GND) to ISPD 2019. This makes routing with cell movement even more challenging. Figure 5.9 shows two possible scenarios where a special net in the Metal2 layer can block the pin access (Figure 5.9 (a)), or it can cause a spacing violation during the VIA insertion (Figure 5.9 (b)). Therefore, CRP encourages



(a) Placement under Special Nets (b) Placement Close to Special Nets

Figure 5.9: This figure shows different scenarios that pin access violations can happen with the existence of special nets. (a) Placement in the lower layer that overlaps with special nets; (b) Placement close to special nets that can cause a spacing violation.

cells to keep their distance from special nets during the placement. This can ease the process of finding pin access for the cells close to special nets and reduce the runtime in the detailed routing.

5.5.6 Potential for further improvement in results

CRP provides a framework that can improve the detailed routing score by routing with cell movement through iterations. Figure 5.10 illustrates that by iteration the detailed routing score can be improved. Figure 5.10(a) shows the percentage of the cells that are moved in each flow, and in Figure 5.10(b), the percentage difference of the detailed routing score compared with the baseline is reported. It can be observed that a 0.3% improvement in the total detailed routing score can be attained by an average of 0.7% movement in the cells. Although I could achieve better results in most of the benchmarks after iteration, only in 2 benchmarks ISPD 2018 test2, test10 after iteration the improvement could not achieved, and in ISPD 2018 test3 and test4 were minor. This shows the necessity of further advanced techniques in generating candidate placement which is one of our future works.

5.5.7 The Detailed Routing Score of CRP versus ISPD 2018 and 2019 Contest winners

The proposed flow utilizes CUGR [25] as a global router, followed by the CRP engine that integrates routing and placement. TritonRoute [91] is then employed as the detailed routing engine, resulting in a significantly improved detailed routing score compared to the ISPD 2018 [42] and ISPD 2019 [43] contest winners. A comparison of the CRP and ISPD winners' raw detailed routing scores is presented in Table 5.6. The proposed

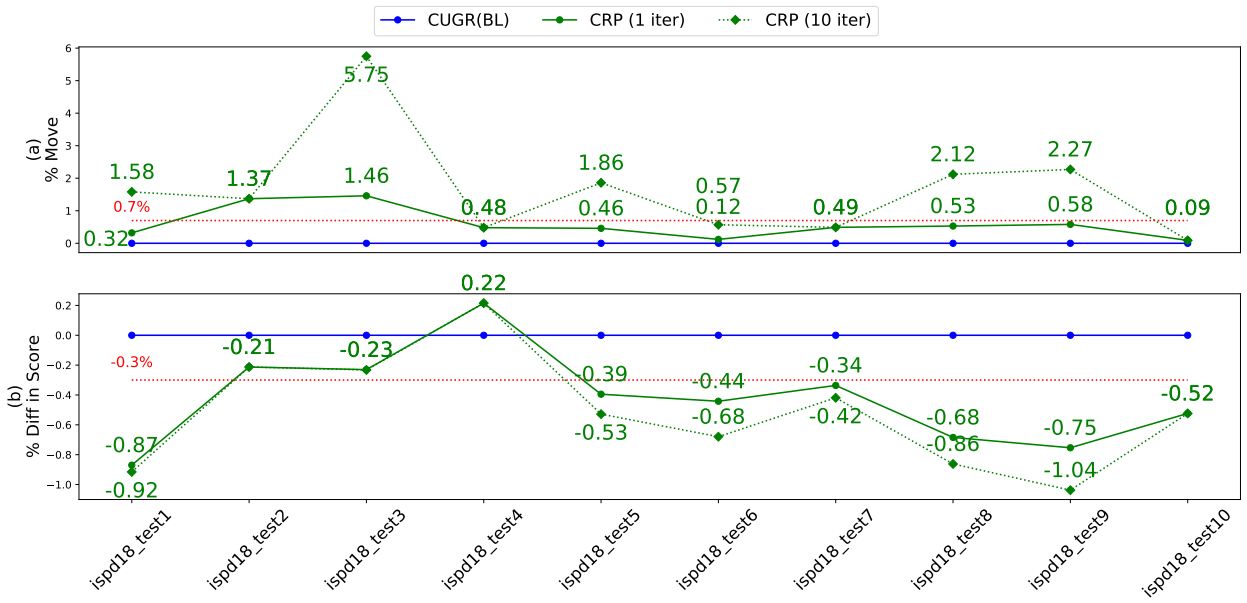


Figure 5.10: CRP can improve the detailed routing score by using iteration. (a) The percentage of cells that are moved. (b) The percentage improvement from the baseline. Note: Negative percentage means improvement.

CRP flow outperforms the ISPD 2018 and 2019 contest winners in all benchmarks, with the exception of ispd19_test4, where the framework encountered failure during the detailed routing step.

Table 5.6: The raw Detailed Routing (DR) score of CRP comparing with ISPD 2018 and ISPD 2019 contest winners.

Benchmarks	1 Place ISPD 2018	CRP	Benchmarks	1 Place ISPD 2019	CRP
	DR Score	DR Score		DR Score	DR Score
ispd18_test1	386188	**299131**	**ispd19_test1**	626129	**415328**
ispd18_test2	5636274	**4799436**	**ispd19_test2**	21942353	**14609221**
ispd18_test3	8645534	**5278893**	**ispd19_test3**	1044187	**590283**
ispd18_test4	67978777	**14895476**	**ispd19_test4**	**22755592**	Failed
ispd18_test5	65227108	**15562038**	**ispd19_test5**	3301288	**2827620**
ispd18_test6	89303690	**20536284**	**ispd19_test6**	47916252	**37526449**
ispd18_test7	Failed	**36856186**	**ispd19_test7**	99164797	**70512672**
ispd18_test8	161426598	**37168218**	**ispd19_test8**	137211940	**107670877**
ispd18_test9	144221466	**31623654**	**ispd19_test9**	216753037	**164918127**
ispd18_test10	193867714	**39459521**	**ispd19_test10**	216333270	**163498327**

5.6 Runtime Analysis

CRP exploits two main techniques to improve the runtime of routing with cell movement. These techniques, Cost Caching and Net Caching, are described in Section 4.4. In this chapter, the runtime improvement due to the Cost and Net Caching techniques on the benchmark circuits are presented first. Next, the breakdown of the runtime and scalability of CRP are discussed.

5.6.1 Runtime Improvement by Cost Caching

The runtime of PatternRoute in CUGR to CRP are compared in Figure 5.11, with Figure 5.11(a) displaying the runtime of routing all nets. Figure 5.11(b) shows the percentage improvement achieved in CRP compared to CUGR PatternRoute through the use of the cost cache technique, resulting in an average reduction of 28.56% in runtime. This improvement is due to the Cost Caching technique avoiding costly recalculations in PatternRoute. Additionally, Figure 5.12 illustrates the increased average speedup for nets with more pins, thanks to the Cost Caching technique. The average speedup for nets with less than 30 pins is 1.8% and for nets with more than 30 pins is 2.6%. These results demonstrate the effectiveness of the proposed cost caching method in improving the runtime of global routing.

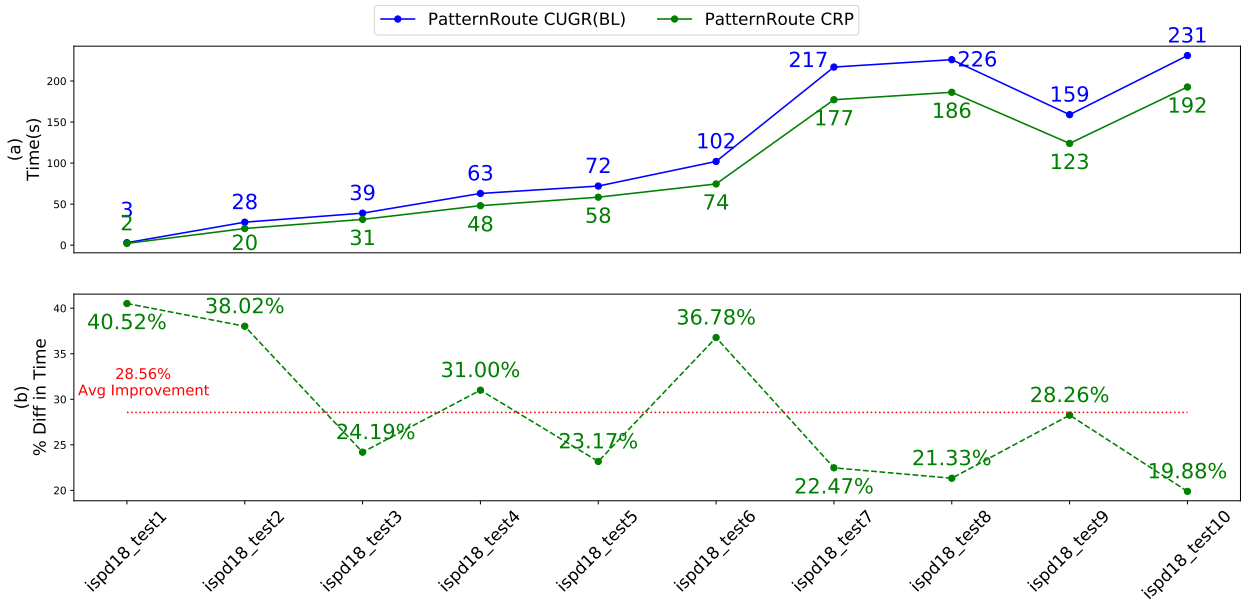


Figure 5.11: This Figure shows the effect of cost caching on the PatternRoute. (a) The runtime of routes of all nets by PatternRoute with (CRP) and without(CUGR(BL)) cost caching. (b) The percentage of runtime's improvement by using cost caching.

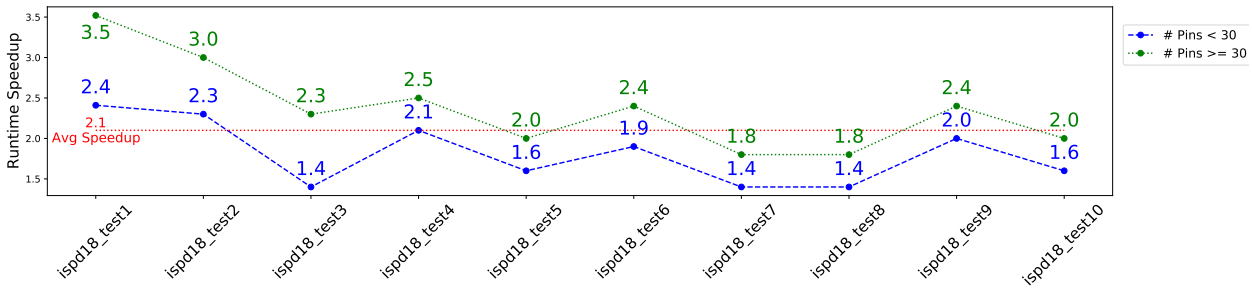


Figure 5.12: The effects of cost caching technique on the nets with a different number of pins. In this figure, the speed-up of the global routing in CRP compared with CUGR for nets with less than 30 pins and greater than or equal to 30 pins is illustrated.

5.6.2 CRP Runtime Analysis in Single and 8 Cores CPU

In Figure 5.13a and Figure 5.13b, the percentage runtime breakdown of CRP running on single and eight cores CPUs are reported. In these figures, the CRP runtime is divided into Global Routing (GR), Label Candidate Cells (LCC), Generate Candidate Positions (GCP), Estimate Candidate Cost (ECC), Find Best Candidates (FBC), and Update Database (UD). According to Figure 5.13a, the second major runtime usage after the global routing is the ECC step. This shows that the quality evaluation of each placement by the 3D global routing is a time-consuming task. This makes it necessary to use parallel computation methods. Therefore, the Net Caching technique (described in Subsection 4.4.2) is proposed to parallelize routing with cell movement. The speedup of the CRP flow after running in 8 cores is reported in Figure 5.14. It can be seen that in ISPD 2019 less speedup was achieved due to the complexity of the circuits after using parallel computation. On average, the total runtime is improved by 2.6 times in all circuits.

The CRP engine was unable to achieve 8 times faster runtime despite being parallelized and run on 8 cores. This can be attributed to Amdahl's Law [120], which states that the overall speedup of a parallelized system is limited by the proportion of the program that cannot be parallelized. In simpler terms, the maximum speedup that can be achieved by parallelizing a program is restricted by the portion of the program that cannot be parallelized.

Amdahl's Law can be expressed using the following formula:

$$S = \frac{1}{(1 - P) + \frac{P}{N}} \tag{5.1}$$

where S is the overall speedup, P is the proportion of the program that can be parallelized, and N is the number of cores.

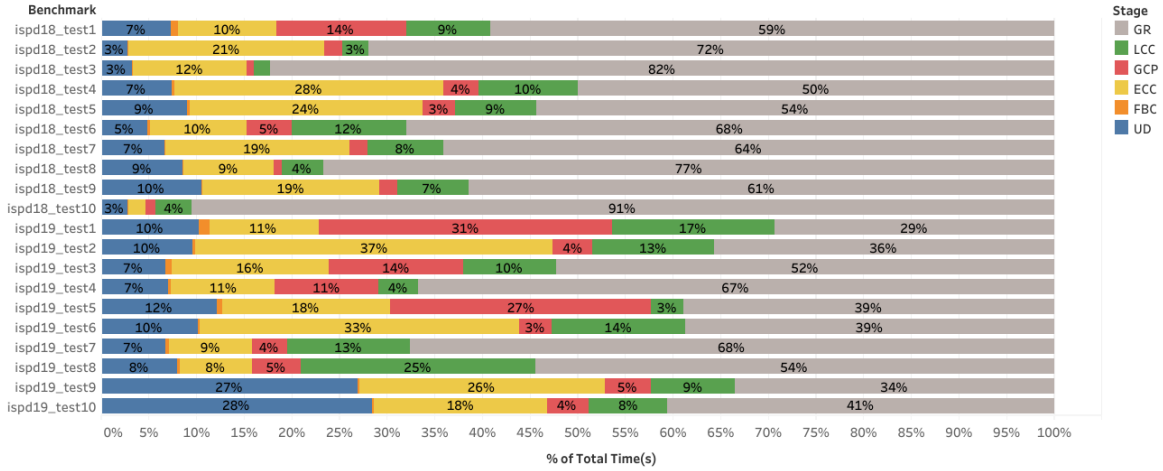
In the case of the CRP engine, a speedup of 2.6 times was achieved using 8 cores. Assuming that 80% of the engine is parallelizable and 20% is not, the theoretical maximum speedup can be calculated using

Amdahl's Law:

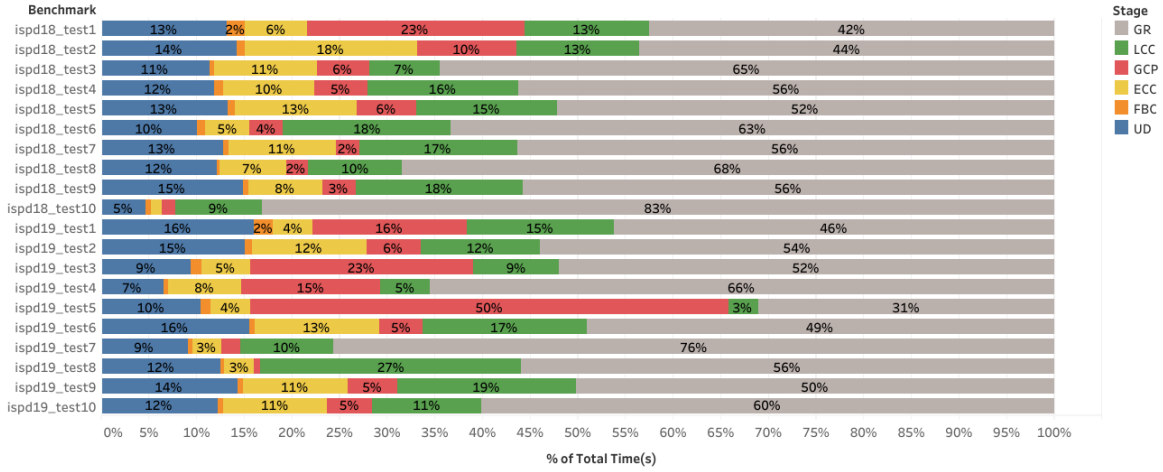
$$S = \frac{1}{(1 - 0.8) + \frac{0.8}{8}} == 3.33 \tag{5.2}$$

This means that the maximum speedup that could be achieved by parallelizing the CRP engine is 3.33 times faster than the original speed. However, the actual speedup of 2.6 times is less than the theoretical maximum. This can be attributed to overhead and communication between the parallel threads, among other factors.

In conclusion, the non-parallelizable portion of the CRP engine and other factors have limited the efficiency of parallelization, preventing it from achieving the maximum possible speedup.



(a) Single Core



(b) 8 cores

Figure 5.13: The percentage of runtime breakdown of CRP running on single and 8 cores CPU for ISPD 2018 and ISPD 2019.

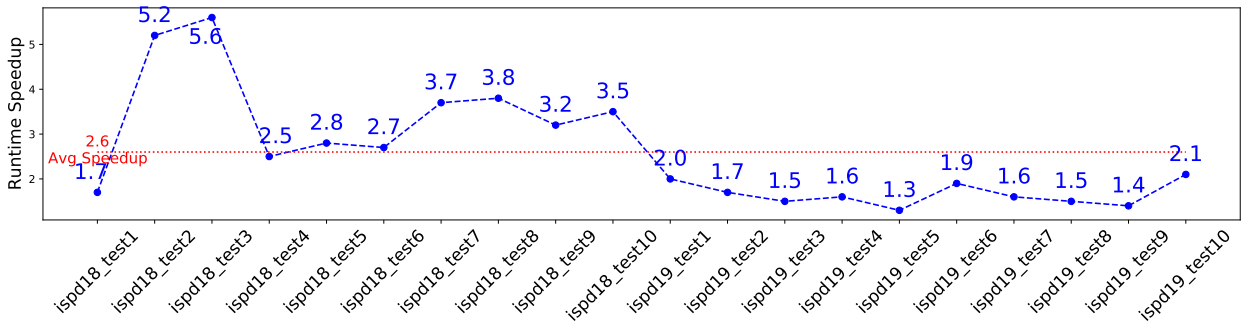


Figure 5.14: The speedup of runtime in CRP flow after running in Parallel (8 Cores) compared with a single core CPU.

5.6.3 CRP Scalability Analysis

In Figure 5.15, the runtime and memory usage of CRP in single and eight cores are illustrated. As can be seen, the proposed CRP can improve the overall runtime in parallel and will not add a huge amount of memory usage that makes this framework scalable in bigger circuits. To avoid memory leakage in CRP, local memories are assigned and released after each process. This can help the CRP engine to avoid a big overhead in memory usage.

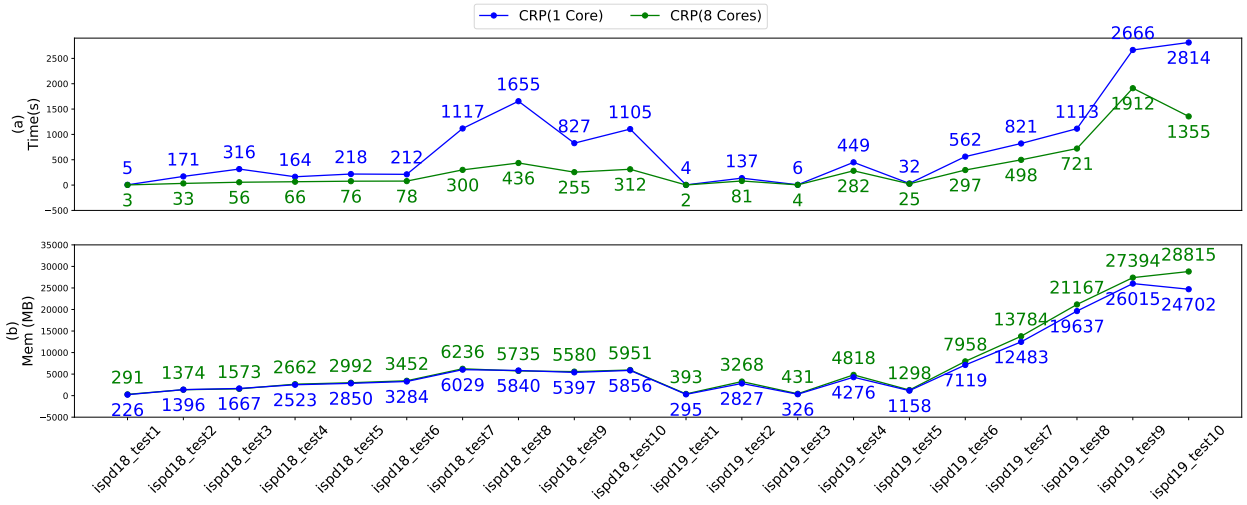


Figure 5.15: This figure shows the scalability of CRP. The runtime and memory usage of CRP on ISPD 2018 and ISPD 2019 benchmarks on 1 and 8 cores CPU reported.

5.6.4 CRP runtime vs other flows

Figure 5.16 compares the total flow runtime (Figure 5.4) and the detailed routing runtime of CRP, CUGR(BL), ABCDPlace, Eh?Placer, and ILP(Mov) running on 8 cores. This figure shows that compared to ABCD-Place, ILP(Mov), and Eh?Placer the total flow of CRP requires less runtime. This can be justified due

to differences between detailed routing runtimes of each flow. Moreover, in ispd18_test10, where the cell utilization is 100% after applying CRP, the detailed routing runtime is improved. Figure 5.17 compares the total flow and detailed routing runtime of CRP vs CUGR(BL). It can be seen in this figure that detailed routing runtime fluctuates and in some benchmarks could achieve runtime improvement like ispd19_test10. However, this fluctuation presents the complexity of the circuit in placement that CRP did not model. This shows the necessity of improvement in CRP for generating candidate placements that can affect the runtime.

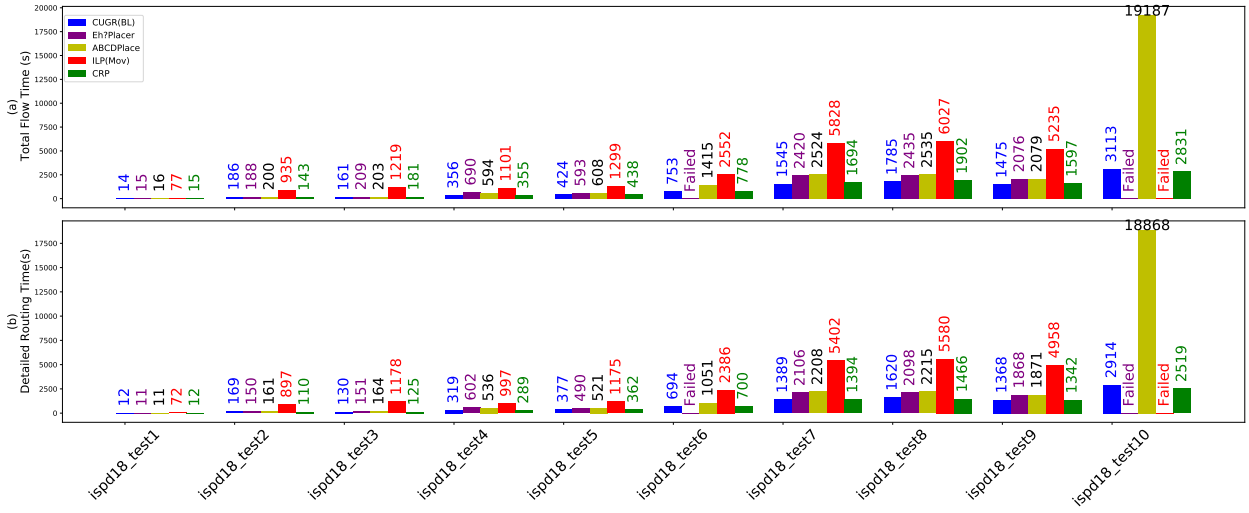


Figure 5.16: This Figure compares the total runtime of the five different flows in the first graph (a) on ISPD 2018. The second graph (b) reports the detailed routing runtime of each flow on ISPD 2018.

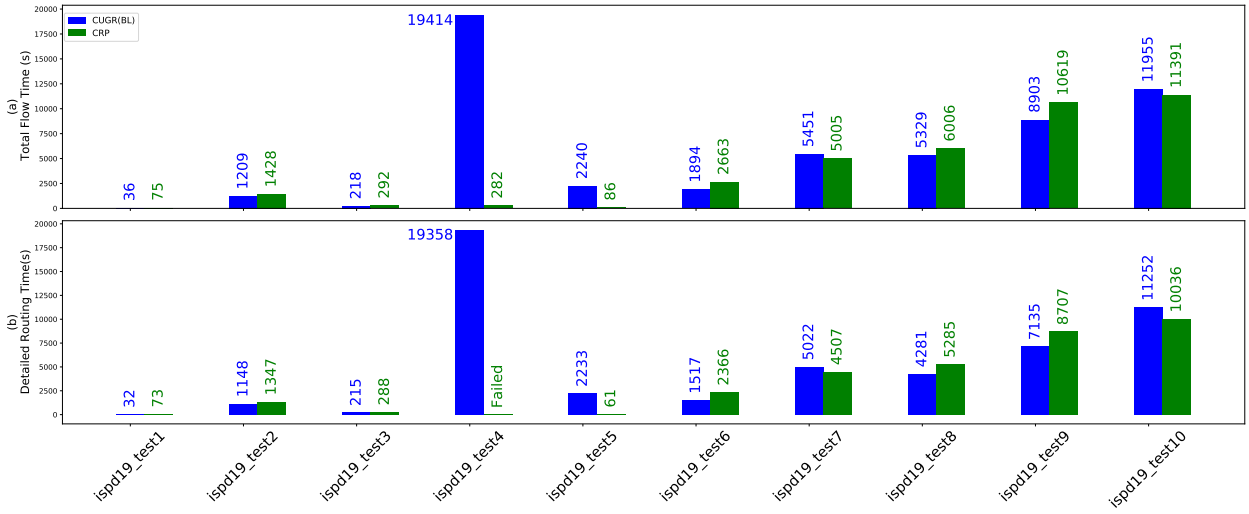


Figure 5.17: This Figure compares the total runtime of the baseline and CRP flows in the first graph on ISPD 2019. The second graph reports the detailed routing runtime of each flow on ISPD 2019.

5.6.5 Overhead of CRP under different iterations

The overhead of CRP under different iterations is presented by comparing the runtime between CUGR (the baseline), CRP (1 iteration), and CRP (10 iterations) in Table 5.7. The effect of cost caching and pruning search technique in ILP-DP on the performance of CRP is also demonstrated by running two versions of CRP: CRPv1 without cost caching and pruning, and CRPv2 with these benefits.

According to this table, the CRP adds around 1.9 times overhead to the CUGR after the first iteration. After 10 iterations, this runtime overhead can increase by 5.2 times CUGR's runtime, on average. Moreover, it can be observed that the runtime overhead in the two biggest benchmarks, ISPD 2018 test8 and 9, is bigger than other benchmarks, around 8.4 times. Also, according to Figure 5.10 (b) for both mentioned benchmarks around 0.2% improvement is achieved through iterations. This means that the size of benchmarks with the number of successful movable cells to reduce the detailed routing score can impact the runtime and result in a bigger runtime overhead. Furthermore, it can be observed in Table 5.7, the runtime overhead of CRP has been reduced by around two times in comparison with CRPv1. There are two key techniques used in CRP to achieve this speedup. First, the runtime of the global router estimator to evaluate candidate placements was reduced using the proposed Cost Caching Technique (Subsection 4.4.1). Second, a pruning search technique (explained in section Subsection 4.3.2) is used in ILP-DP to reduce the dimension of search space which results in a runtime improvement of the ILP-DP method.

Table 5.7: Overhead that the proposed add-on engine CRP has under different iterations. The runtime of CUGR, CRP first iteration, and CRP 10 iteration are shown in columns 2, 3, and 5 and are based on seconds (s). The overhead of the first and tenth iterations are shown in columns 4 and 6. The overhead is calculated by $\frac{(\text{CRP+CUGR})}{\text{CUGR}}$. The effect of cost caching and pruning search technique in ILP-DP on the performance of CRP is also demonstrated by running two versions of CRP: CRPv1 without cost caching and pruning, and CRPv2 with cost caching and pruning search technique.

benchmarks	Runtime (s)	Runtime 1 iteration (s)		Overhead 1 iteration		Runtime 10 iterations		Overhead 10 iterations	
	CUGR	CRPv1	CRPv2	CRPv1	CRPv2	CRPv1	CRPv2	CRPv1	CRPv2
ispd18_test1	4.6	5.2	2.6	2.1	1.6	22.0	11.7	5.8	3.5
ispd18_test2	39.9	72.3	31.0	2.8	1.8	463.5	164.6	12.6	5.1
ispd18_test3	55.8	106.3	52.9	2.9	1.9	628.5	243.4	12.3	5.4
ispd18_test4	87.0	115.0	61.1	2.3	1.7	661.7	286.2	8.6	4.3
ispd18_test5	102.5	144.7	68.8	2.4	1.7	809.8	321.1	8.9	4.1
ispd18_test6	123.9	124.2	76.4	2.0	1.6	551.0	289.2	5.4	3.3
ispd18_test7	310.8	572.2	295.1	2.8	1.9	3147.6	1260.8	11.1	5.1
ispd18_test8	334.4	706.8	436.6	3.1	2.3	4543.2	2460.7	**14.6**	**8.4**
ispd18_test9	209.5	400.9	254.1	2.9	2.2	2964.8	1540.3	**15.2**	**8.4**
ispd18_test10	285.6	356.2	313.2	2.2	2.1	1177.1	857.1	5.1	4.0
Average	-	-	-	**2.6**	**1.9**	-	-	**10.0**	**5.2**

5.7 Summary

In this chapter, the results of the proposed flow on ISPD 2018 and 2019 benchmarks are presented. The CRP engine outperforms state-of-the-art detailed placement engines and improves detailed routing scores. Also, the importance of the proposed path cost in the improvement of the detailed routing score is illustrated. The potential for further improvement with an iterative approach is also discussed, with numerical results indicating 0.3% improvement in detailed routing score by moving 0.7% of cells on average. The runtime of the CRP engine and the impact of cost and net caching techniques are presented, with results showing a 28.56% improvement in global routing runtime and a 2.6x speedup with parallel processing. The scalability of the proposed flow is analyzed and overall runtime results on ISPD benchmarks are provided, demonstrating the potential effectiveness of the CRP engine as an add-on to global routing.

Chapter 6

conclusions and Future Work

In this thesis, a framework that allows us to implement cooperation between routing and placement is proposed. The main objective of the proposed framework is to improve the detailed routing scores by combining routing and placement. The core of this framework is the CRP engine which includes techniques to combine routing and placement.

The experimental results of CRP compared to CUGR(BL), Eh?Placer, ABCDPlace, and ILP(Mov) were presented. This shows that techniques used in CRP efficiently can improve the detailed routing score of both the ISPD 2018 and ISPD 2019 benchmarks compared to other flows. In addition, due to the inclusion of several techniques proposed to improve the runtime of routing with cell movement, it shows that CRP can affect the detailed routing runtime and it can improve it in some case studies.

The source code for the proposed framework, including the techniques developed in the CRP engine, is open-source and can be accessed at [121]. The main contributions of this thesis are as follows:

- **Proposing and developing a framework that enables cooperation between routing and placement while respecting technology node constraints:** The proposed framework comprises three key components: a Global Routing (GR) engine, a Cooperation between Routing and Placement (CRP) engine, and a Detailed Routing (DR) engine.
- **Proposing an efficient cooperation engine between routing and placement called CRP:** This engine can be used to fill the gap between the placement and routing steps to empower the routing solution. The five main steps are proposed and developed in the CRP engine:
  - **Label Candidate Cells (LCC):** In this step, candidate cells for movement are tagged according to their initial routing path cost.

- **Generate Candidate Positions (GCP):** In this step, a list of potential cell placements is generated for cells previously tagged.
- **Estimate Candidate Cost (ECC):** The 3D routing cost of each potential cell placement generated in the GCP step is calculated in this step.
- **Find Best Candidates (FBC):** In this step, the best potential cell placements are chosen based on their 3D routing cost.
- **Update Database (UD):** In this step, the database is updated by relocating the chosen cells and reroute them.

- **Developing an ILP-based Detailed Placement (ILP-DP):** A new ILP-DP is introduced to generate legal placement options for moving cells. A new cost function called Congestion-aware Moving-Toward Cell's Median (CM-TCM) is also proposed to select the optimal placement option for evaluation by 3D global routing.

- **Proposing two modes in global routing:** The proposed two modes in global routing are: Fixed Placement Mode (FPM) and Rip-up Placement Mode (RPM). A caching technique called Cost Caching is introduced to improve the efficiency of global routing, reducing runtime by 28.56% compared to the state-of-the-art. A Net Caching technique is also proposed for parallel execution of cooperation between routing and placement, resulting in a 2.6 times speedup compared to sequential methods. A net classification technique is used to determine the routing algorithm for each cell movement, and two global routing algorithms, AStar and PatternRoute, are used to calculate the path cost for each cell location.

- **Demonstrating the effectiveness of the proposed techniques on benchmarks that include detailed routing information (ISPD 2018 and ISPD 2019):** Evaluating the results in these benchmark suites enables us to investigate the quality of proposed techniques on the detailed routing metrics.

In summary, the difficulties encountered when routing with cell movement in advanced technology nodes and some techniques used to address them include:

- The optimization and objective goals can conflict with each other. For example, it is common to improve HPWL in placement engines while putting a big weight on HPWL and absolute wirelength improvement can increase congested areas. This can result in a degradation in the detailed routing score. This is shown in Figure 5.5 and Figure 5.8.

- The complexities imposed by advanced technology nodes require specification in both flow and techniques used to improve the detailed routing score. Figure 5.9 illustrates the effects of special nets in the placement of cells. Also, Figure 5.6 illustrates the necessity of using different routing algorithms according to net features inside the layout.

- Routing with cell movement due to high dependency is a sequential flow and is hard to parallelize. One way to parallelize this flow is to use the proposed caching techniques. Also, to avoid memory usage overhead the memory assignment should be local and released after each process.

- Using techniques like the ILP model can be very useful in detailed placement when there is a small search space with constraints. However, if the search space increases the ILP model can be slow. This is shown in Figure 5.13b ispd19_test5, where the placement density is around 9%.

6.1 Future Work

Based on the key findings of this thesis, two potential directions for future research are proposed:

- **Accelerating the CRP engine with GPU:** Recent parallel and heterogeneous techniques can be used to improve the CRP runtime. The current version of CRP is implemented in the CPU and uses multi-threading for parallelism. To accelerate this, the workload in the CRP engine can be decomposed into CPU-GPU-dependent tasks. This idea draws inspiration from [122], where normal or low computation tasks are powered by the CPU and massive tasks are assigned to a GPU. For example, in the ILP-DP step of the CRP engine, a GPU can be used as an accelerator to generate alternative placement candidates. Techniques proposed by ABCDPlace [37] can be exploited to use the ILP-DP step. Moreover, thanks to the Cost Net caching proposed in this thesis, a combination of the routing and placement processes can be parallelized. Also, the deterministic techniques introduced in [123] can be used to further accelerate the evaluation of the quality of each placement by routing.

- **Cooperation between global and detailed routing with detailed placement:** This thesis explores the impact of integrating global routing and detailed placement on improving the detailed routing solution quality. However, a strong dependence on global routing solution quality is noted as a weakness of this approach. If the correlation between global and detailed routing is poor, it can result in decreased quality of the detailed routing solution. Thus, combining both global and detailed routing with detailed placement is a promising area for further research.

- **Leveraging CRP to other physical design steps:** The potential of the CRP engine to extend to

other steps of the physical design depends on how the objective function is defined, as presented in Figure 5.8, which shows how different objectives can impact the final routing solution. In this thesis, the detailed routing score is addressed as a case study. However, there are other parts of the physical design, such as time-driven routing, that could be potential areas for future work of this thesis.

Bibliography

[1] A. B. Kahng, J. Lienig, I. Markov, and J. Hu, *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Springer, 2011th. ed., Jan 2011.

[2] P. W. Jacobi, "Semiconductor amplifier," Germany Patent DE833366C, April. 1949.

[3] J. Kilby, "Miniaturized electronic circuits," United States Patent US3138743A, Feb. 1959.

[4] N. Aeronautics and S. Administration, "James webb telescope." `https://jwst.nasa.gov/content/multimedia/images.html`, 2022.

[5] S. Cass, "Chip hall of fame: Intel 4004 microprocessor." `https://spectrum.ieee.org/chip-hall-of-fame-intel-4004-microprocessor`, 2018.

[6] V. Chakravarthi, *A Practical Approach to VLSI System on Chip (SoC) Design*. Springer International Publishing AG, 2022.

[7] A. A. Jazie, A. J. Albaaji, and S. A. Abed, "A review on recent trends of antiviral nanoparticles and airborne filters: special insight on covid-19 virus.," *Air Qual Atmos Health*, no. 14, p. 21811–1824, 2021.

[8] TSMC, "Logic technology at taiwan semiconductor manufacturing company(tsmc)." `https://www.tsmc.com/english/dedicatedFoundry/technology/logic`, 2020.

[9] Synopsys, "Synopsys, inc.." `https://www.synopsys.com/`, Mountain View, California, United States, 1986.

[10] Cadence, "Cadence design systems, inc.." `https://www.cadence.com/en_US/home.html`, San Jose, California, United States, 1988.

[11] A. B. Kahng, "The itrs design technology and system drivers roadmap: Process and status," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2013.

[12] T. N. Bui, C. Heigham, C. A. Jones, and T. Leighton, "Improving the performance of the kernighan-lin and simulated annealing graph bisection algorithms," *Proceedings of the 26th ACM/IEEE Design Automation Conference*, vol. 67, no. 4, p. 775–778, 1989.

[13] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: applications in vlsi domain," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, pp. 69–79, 1999.

[14] C. Kyung, J. Widder, and D. Mlynski, "Adaptive cluster growth (acg); a new algorithm for circuit packing in rectilinear region," in *Proceedings of the European Design Automation Conference*, pp. 191–195, IEEE, 1990.

[15] T.-C. Chen and Y.-W. Chang, "Modern floorplanning based on b/sup */-tree and fast simulated annealing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 4, pp. 637–650, 2006.

[16] M. Lapan, *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing, 1 ed., June 2018.

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 1 ed., November 2016.

[18] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi1, J. Pak, A. Tong, K. Srinivasa, W. Hang, E. Tuncer, Q. V. Le, J. Laudon, R. Ho, R. Carpenter, and J. Dean, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 1, pp. 207–2012, 2021.

[19] A. Mirhoseini, "Circuit training: An open-source framework for generating chip floor plans with distributed deep reinforcement learning.." `https://github.com/google-research/circuit_training`, 2021.

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

[21] R. Netto, S. Fabre, T. A. Fontana, V. Livramento, L. L. Pilla, L. Behjat, and J. L. Güntzel, "Algorithm selection framework for legalization using deep convolutional neural networks and transfer learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 5, p. 1481–1494, 2022.

[22] H. Yang, K. Fung, Y. Zhao, Y. Lin, and B. Yu, "Mixed-cell-height legalization on cpu-gpu heterogeneous systems," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 784–789, IEEE, 2022.

[23] C. Wu, W. K. Mak, and C. C. N. Chu, "Linear-time mixed-cell-height legalization for minimizing maximum displacement," in *Proceedings of the 2022 International Symposium on Physical Design*, p. 211–218, Association for Computing Machinery, 2022.

[24] Y. Xu, Y. Zhang, and C. Chu, "Fastroute 4.0: Global router with efficient via minimization," in *2009 Asia and South Pacific Design Automation Conference*, IEEE, 2009.

[25] J. Liu, C.-W. Pui, F. Wang, and E. F. Y. Young, "Cugr: Detailed-routability-driven 3d global routing with probabilistic resource model," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2020.

[26] A. B. Kahng, L. Wang, and B. Xu, "Tritonroute-wxl: The open-source router with integrated drc engine," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 4, pp. 1076–1089, 2022.

[27] C. J. Alpert, D. P. Mehta, and S. S. Sapatnekar, *Handbook of Algorithms for Physical Design Automation*. Auerbach Publications, 1st. ed., November 2008.

[28] W.-H. Liu, W.-C. Kao, Y.-L. Li, and K.-Y. Chao, "Nctu-gr 2.0: Multithreaded collision-aware global routing with bounded-length maze routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 5, pp. 709–722, 2013.

[29] Y.-J. Chang, Y.-T. Lee, and T.-C. Wang, "Nthu-route 2.0: A fast and stable global router," in *2008 IEEE/ACM International Conference on Computer-Aided Design*, IEEE, 2008.

[30] B. Lin, F. Zhu, J. Zhang, J. Chen, X. Chen, N. N. Xiong, and J. L. Mauri, "A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 4254–4265, 2019.

[31] M. Baldi and G. Marchetto, "Time-driven priority router implementation: Analysis and experiments," *IEEE Transactions on Computers*, vol. 62, no. 5, pp. 1017–1030, 2013.

[32] L. Josipović, S. Sheikhha, A. Guerrieri, P. Ienne, and J. Cortadella, "Buffer placement and sizing for high-performance dataflow circuits," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 1, p. 32, 2021.

[33] A. B. Kahng, "Advancing placement," in *ISPD 2021: Proceedings of the 2021 International Symposium on Physical Design*, (New York, NY, USA), pp. 15–22, Association for Computing Machinery, 2021.

[34] K.-S. Hu, M.-J. Yang, T.-C. Yu, and G.-C. Chen, "Iccad-2020 cad contest in routing with cell movement : Invited talk," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–4, IEEE, 2020.

[35] K.-S. Hu, T.-C. Yu, M.-J. Yang, and C.-F. C. Shen, "2021 iccad cad contest problem b: Routing with cell movement advanced: Invited paper," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–5, IEEE, 2021.

[36] N. K. Darav, A. A. Kennings, A. F. Tabrizi, D. T. Westwick, and L. Behjat, "Eh?placer: A high-performance modern technology-driven placer," *ACM Transactions on Design Automation of Electronic Systems*, vol. 21, no. 3, pp. 1–27, 2016.

[37] Y. Lin, W. Li, J. Gu, H. Ren, B. Khailany, and D. Z. Pan, "Abcdplace: Accelerated batch-based concurrent detailed placement on multithreaded cpus and gpus," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 5083–5096, 2020.

[38] M. Pan and C. Chu, "Ipr: An integrated placement and routing algorithm," in *2007 44th ACM/IEEE Design Automation Conference*, pp. 59–62, IEEE, 2007.

[39] J. A. Roy, N. Viswanathan, G.-J. Nam, C. J. Alpert, and I. L. Markov, "Crisp: Congestion reduction by iterated spreading during placement," in *2009 IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers*, pp. 357–362, IEEE, 2009.

[40] H. Shojaei, A. Davoodi, and J. T. Linderoth, "Congestion analysis for global routing via integer programming," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 256–262, IEEE, 2011.

[41] Y. Wei, C. Sze, N. Viswanathan, Z. Li, C. J. Alpert, L. Reddy, A. D. Huber, G. E. Tellez, D. Keller, and S. S. Sapatnekar, "Glare: Global and local wiring aware routability evaluation," in *DAC Design Automation Conference 2012*, pp. 768–773, IEEE, 2012.

[42] S. Mantik, G. Posser, W. K. Chow, Y. Ding, and W.-H. Liu, "Ispd 2018 initial detailed routing contest and benchmarks," in *ISPD 2018: Proceedings of the 2021 International Symposium on Physical Design*, (Monterey, California, US), pp. 140–143, Association for Computing Machinery, 2018.

[43] W.-H. Liu, S. Mantik, W. K. Chow, Y. Ding, A. Farshidi, and G. Posser, "Ispd 2019 initial detailed routing contest and benchmark with advanced routing rules," in *ISPD 2019: Proceedings of the 2021 International Symposium on Physical Design*, (New York, NY, USA), pp. 147–151, Association for Computing Machinery, 2019.

[44] E. Aghaeekiasaraee, A. F. Tabrizi, T. A. Fontana, R. Netto, S. F. Almeida, U. Gandhi, J. L. Güntzel, D. Westwick, and L. Behjat, "Crp: An efficient co-operation between routing and placement," in *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 772–777, Springer New York, 2022.

[45] E. Aghaeekiasaraee, A. F. Tabrizi, T. A. Fontana, R. Netto, S. F. Almeida, U. Gandhi, J. L. Güntzel, D. Westwick, and L. Behjat, "Crp2.0: A fast and robust cooperation between routing and placement in advanced technology nodes," *ACM Trans. Des. Autom. Electron. Syst.*, 2023. Just Accepted.

[46] C. D. Systems, "Lef/def language reference." `https://www.ispd.cc/contests/18/lefdefref.pdf`, 2009.

[47] J. Chen, Z. Zhu, W. Zhu, and Y.-W. Chang, "Toward optimal legalization for mixed-cell-height circuit designs," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2017.

[48] Synopsys, "Lef/def parser." `https://si2.org/oa-tools-utils-libs/`.

[49] G. Flach, M. Fogaça, J. Monteiro, M. O. Johann, and R. A. L. Reis, "Rsyn: An extensible physical synthesis framework," in *Proceedings of the 2017 ACM on International Symposium on Physical Design*, p. 33–40, Association for Computing Machinery, 2017.

[50] T. Taghavi, X. Yang, and B. Choi, "Dragon2005: Large-scale mixed-size placement tool," in *Proceedings of the 2005 International Symposium on Physical Design*, p. 245–247, Association for Computing Machinery, 2005.

[51] T. Chen, T. Hsu, Z. Jiang, and Y.-W. Chang, "Ntuplace: A ratio partitioning based placement algorithm for large-scale mixed-size designs," in *Proceedings of the 2005 International Symposium on Physical Design*, p. 236–238, Association for Computing Machinery, 2005.

[52] I. L. Markov, J. Hu, and M.-C. Kim, "Progress and challenges in vlsi placement research," *Proceedings of the IEEE*, vol. 103, no. 11, pp. 1985–2003, 2015.

[53] J. Lu, P. Chen, C. Chang, L. Sha, D. J. H. Huang, C. Teng, and C.-K. Cheng, "Eplace: Electrostatics-based placement using fast fourier transform and nesterov's method," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 20, no. 2, pp. 1–34, 2015.

[54] C. Sechen and A. L. Sangiovanni-Vincentelli, "The timberwolf placement and routing package," *IEEE Journal of Solid-State Circuits*, vol. 20, no. 2, pp. 510–522, 1985.

[55] T. Lin, C. Chu, J. R. Shinnerl, I. Bustany, and I. Nedelchev, "Polar: A high performance mixed-size wirelengh-driven placer with density constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 3, pp. 447–459, 2015.

[56] X. He, T. Huang, W.-K. Chow, J. Kuang, K.-C. Lam, W. Cai, and E. F. Young, "Ripple 2.0: High quality routability-driven placement via global router integration," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2013.

[57] C.-C. Huang, H.-Y. Lee, B.-Q. Lin, S.-W. Yang, C.-H. Chang, S.-T. Chen, and Y.-W. Chang, "Ntuplace4dr: A detailed-routing-driven placer for mixed-size circuit designs with technology and region constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 3, pp. 669–681, 2018.

[58] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "Dreampiace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2019.

[59] F. Gessler, P. Brisk, and M. Stojilovič, "A shared-memory parallel implementation of the replace global cell placer," in *2020 33rd International Conference on VLSI Design and 2020 19th International Conference on Embedded Systems (VLSID)*, pp. 78–83, IEEE, 2020.

[60] P. Spindler, U. Schlichtmann, and F. M. Johannes, "Abacus: Fast legalization of standard cell circuits with minimal movement," in *Proceedings of the 2008 International Symposium on Physical Design*, p. 47–53, Association for Computing Machinery, 2008.

[61] W. K. Chow, J. Kuang, X. He, W. Cai, and E. F. Y. Young, "Cell density-driven detailed placement with displacement constraint," in *Proceedings of the 2014 on International Symposium on Physical Design*, p. 3–10, Association for Computing Machinery, 2014.

[62] Y. Lin, B. Yu, X. Xu, J.-R. Gao, N. Viswanathan, W.-H. Liu, Z. Li, C. J. Alpert, and D. Z. Pan, "Mrdp: Multiple-row detailed placement of heterogeneous-sized cells for advanced nodes," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2016.

[63] J. Chen, Y.-W. Chang, and Y.-Y. Wu, "Mixed-cell-height detailed placement considering complex minimum-implant-area constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 10, pp. 2128–2141, 2021.

[64] N. Viswanathan and C. C.-N. Chu, "Fastplace: efficient analytical placement using cell shifting, iterative local refinement,and a hybrid net model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 5, pp. 722–733, 2005.

[65] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "Ntuplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1228–1240, 2008.

[66] M.-K. Hsu, Y.-F. Chen, C.-C. Huang, T.-C. Chen, and Y.-W. Chang, "Routability-driven placement for hierarchical mixed-size circuit designs," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, p. 1–6, IEEE, 2013.

[67] N. Viswanathan, M. Pan, and C. Chu, "Fastplace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control," in *2007 Asia and South Pacific Design Automation Conference*, p. 135–140, IEEE, 2007.

[68] M. Pan, N. Viswanathan, and C. Chu, "An efficient and effective detailed placement algorithm," in *IEEE/ACM International Conference on Computer-Aided Design*, p. 48–55, IEEE, 2005.

[69] S. Li and C. Koh, "Mixed integer programming models for detailed placement," in *Proceedings of the 2012 ACM International Symposium on International Symposium on Physical Design*, p. 87–94, Association for Computing Machinery, 2012.

[70] S. Banerjee, Z. Li, and S. R. Nassif, "Iccad-2013 cad contest in mask optimization and benchmark suite," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, p. 271–274, Association for Computing Machinery, 2013.

[71] K.-R. Dai, C.-H. Lu, and Y.-L. Li, "Grplacer: Improving routability and wire-length of global routing with circuit replacement," in *2009 IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers*, pp. 351–356, IEEE, 2009.

[72] M.-C. Kim, J. Hu, D.-J. Lee, and I. L. Markov, "A simplr method for routability-driven placement," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 67–73, IEEE, 2011.

[73] C. Wang, C. Huang, S. S. Y. Liu, C.-Y. Chin, S. Hu, W. Wu, and H.-M. Chen, "Closing the gap between global and detailed placement: Techniques for improving routability," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, p. 149–156, Association for Computing Machinery, 2015.

[74] J. Cong, G. Luo, K. Tsota, and B. Xiao, "Optimizing routability in large-scale mixed-size placement," in *2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 441–446, IEEE, 2013.

[75] M.-K. Hsu, S. Chou, T.-H. Lin, and Y.-W. Chang, "Routability-driven analytical placement for mixed-size circuit designs," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 80–84, IEEE, 2011.

[76] C.-C. Huang, H.-Y. Lee, B.-Q. Lin, S.-W. Yang, C.-H. Chang, S.-T. Chen, and Y.-W. Chang, "Detailed-routability-driven analytical placement for mixed-size designs with technology and region constraints," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 508–513, IEEE, 2015.

[77] W.-H. Liu, C.-K. Koh, and Y.-L. Li, "Optimization of placement solutions for routability," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–9, IEEE, 2013.

[78] T.-H. Wu, A. Davoodi, and J. T. Linderoth, "Grip: Scalable 3d global routing using integer programming," in *2009 46th ACM/IEEE Design Automation Conference*, pp. 320–325, IEEE, 2009.

[79] Y. Xu and C. Chu, "Mgr: Multi-level global router," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 250–255, IEEE, 2011.

[80] C. Chu and Y.-C. Wong, "Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 70–83, 2007.

[81] L. S.-E. David and kim Dae Hyun, "Construction of all rectilinear steiner minimum trees on the hanan grid and its applications to vlsi design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 6, pp. 1165–1176, 2020.

[82] Y. Zhang and C. Chu, "Construction of all rectilinear steiner minimum trees on the hanan grid and its applications to vlsi design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 9, pp. 1655–1668, 2013.

[83] X. Jia, Y. Cai, Q. Zhou, G. Chen, Z. Li, and Z. Li, "Mcfroute: A detailed router based on multi-commodity flow method," in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 397–404, IEEE, 2014.

[84] X. Jia, Y. Cai, Q. Zhou, and B. Yu, "Mcfroute 2.0: A redundant via insertion enhanced concurrent detailed router," in *2016 International Great Lakes Symposium on VLSI (GLSVLSI)*, pp. 87–92, IEEE, 2016.

[85] T. Edwards, "Qrouter." `https://github.com/RTimothyEdwards/qrouter`, 2017.

[86] G. Chen, C.-W. Pui, H. Li, J. Chen, B. Jiang, and E. F. Y. Young, "Detailed routing by sparse grid graph and minimum-area-captured path search," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, p. 754–760, Association for Computing Machinery, 2019.

[87] G. Chen, C.-W. Pui, H. Li, and E. F. Y. Young, "Dr. cu: Detailed routing by sparse grid graph and minimum-area-captured path search," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 9, pp. 1902–1915, 2020.

[88] S. M. M. Gonçalves, L. S. Rosa, and F. S. Marques, "Draps: A design rule aware path search algorithm for detailed routing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 7, pp. 1239–1243, 2020.

[89] A. B. Kahng, L. Wang, and B. Xu, "Tritonroute: An initial detailed router for advanced vlsi technologies," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2018.

[90] F.-K. Sun, H. Chen, C.-Y. Chen, C.-H. Hsu, and Y.-W. Chang, "A multithreaded initial detailed routing algorithm considering global routing guides," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, IEEE, 2018.

[91] A. B. Kahng, L. Wang, and B. Xu, "Tritonroute: The open-source detailed router," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 3, pp. 547–559, 2021.

[92] A. B. K. team, "Tritonroute." `https://github.com/The-OpenROAD-Project/TritonRoute`, 2018.

[93] F. Wang, L. Liu, J. Chen, J. Liu, X. Zang, and M. D. Wong, "Starfish: An efficient p amp;r co-optimization engine with a-based partial rerouting," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, IEEE, 2021.

[94] Z. Huang, H. Huang, R. Shi, X. Li, X. Zhang, W. Chen, and J. Wang, "Detailed placement and global routing co-optimization with complex constraints," *Electronics*, vol. 11, no. 1, 2022.

[95] P. Zou, Z. Cai, Z. Lin, C. Ma, J. Yu, and J. Chen, "Incremental 3d global routing considering cell movement and complex routing constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2022.

[96] X. He, W. K. Chow, and E. F. Y. Young, "Srp: Simultaneous routing and placement for congestion refinement," in *Proceedings of the 2013 ACM International Symposium on Physical Design*, pp. 108–113, Association for Computing Machinery, 2013.

[97] O. team, "Ophidian: Open-source library for physical design research and teaching." `https://github.com/eclufsc/ophidian`, 2018.

[98] T. A. Fontana, E. Aghaeekiasaraee, R. Netto, S. F. Almeida, U. Gandhi, A. F. Tabrizi, D. Westwick, L. Behjat, and J. L. Güntzel, "Ilp-based global routing optimization with cell movements," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 25–30, IEEE, 2021.

[99] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, 2nd ed., 2001.

[100] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis, *R-Trees: Theory and Applications*. Springer London, year =.

[101] A. Guttman, "R-trees: A dynamic index structure for spatial searching," *Special Interest Group on Management of Data*, vol. 14, no. 2, pp. 47–57, 1984.

[102] B. Schling, *The Boost C++ Libraries*. XML Press, 1.68 ed., 2011.

[103] T.-H. Lee and T.-C. Wang, "Congestion-constrained layer assignment for via minimization in global routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 9, pp. 1643–1656, 2008.

[104] S. Liu, Y. Pu, P. Liao, H. Wu, R. Zhang, Z. Chen, W. Lv, Y. Lin, and B. Yu, "Fastgr: Global routing on cpu-gpu with heterogeneous task graph scheduler," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2022.

[105] G. Posser, E. F. Young, S. Held, Y.-L. Li, and D. Z. Pan, "Challenges and approaches in vlsi routing," in *Proceedings of the 2022 International Symposium on Physical Design*, p. 185–192, Association for Computing Machinery, 2022.

[106] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[107] H. Karloff, *The Simplex Algorithm*. Birkhäuser Boston, 2009.

[108] J. C. Nash, "The (dantzig) simplex method for linear programming," *IEEE Educational Activities Department*, vol. 2, no. 1, pp. 29–31, 2000.

[109] P. B. Nyiam and A. Salhi, "On the simplex, interior-point and objective space approaches to multiobjective linear programming," *Journal of Algorithms & Computational Technology*, vol. 15, no. 1, 2021.

[110] Cplex and I. ILOG, "V12. 1: User's manual for cplex," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.

[111] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual." `https://www.gurobi.com`, 2022.

[112] D. B. Johnson, "A note on dijkstra's shortest path algorithm," *Association for Computing Machinery*, vol. 20, no. 3, pp. 385–388, 1973.

[113] M. Moffitt, "Maizerouter: Engineering an effective global router," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, no. 7, pp. 2017–2026, 2008.

[114] M. Pan and C. Chu, "Fastroute: A step to integrate global routing into placement," in *Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design*, p. 464–471, Association for Computing Machinery, 2006.

[115] M. Pan, N. Viswanathan, and C. Chu, "An efficient and effective detailed placement algorithm," in *ICCAD-2005. IEEE/ACM International Conference on Computer-Aided Design, 2005.*, pp. 48–55, IEEE, 2005.

[116] T.-H. Lee and T.-C. Wang, "ongestion-constrained layer assignment for via minimization in global routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 9, pp. 1643–1656, 2008.

[117] T. Lee and T.-C. Wang, "Robust layer assignment for via optimization in multi-layer global routing," in *Proceedings of the 2009 International Symposium on Physical Design*, pp. 159–166, Association for Computing Machinery, 2009.

[118] I. Bustany, D. G. Chinnery, J. R. Shinnerl, and V. Yutsis, "Ispd 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement," in *Proceedings of the 2015 Symposium*

on International Symposium on Physical Design, p. 157–164, Association for Computing Machinery, 2015.

[119] Y. Lin, "Dreamplace." `https://github.com/limbo018/DREAMPlace`, 2022.

[120] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities, reprinted from the afips conference proceedings, vol. 30 (atlantic city, n.j., apr. 18–20), afips press, reston, va., 1967, pp. 483–485, when dr. amdahl was at international business machines corporation, sunnyvale, california," *IEEE Solid-State Circuits Society Newsletter*, vol. 12, no. 3, pp. 19–20, 2007.

[121] E. Aghaeekiasaraee, "Crp2.0: An efficient cooperation between routing and placement with technology node constraints." `https://github.com/erfanAghaee/CRP2.git`, 2022.

[122] Z. Guo, T. Huang, and Y. Lin, "Gpu-accelerated static timing analysis," in *Proceedings of the 39th International Conference on Computer-Aided Design*, Association for Computing Machinery, 2020.

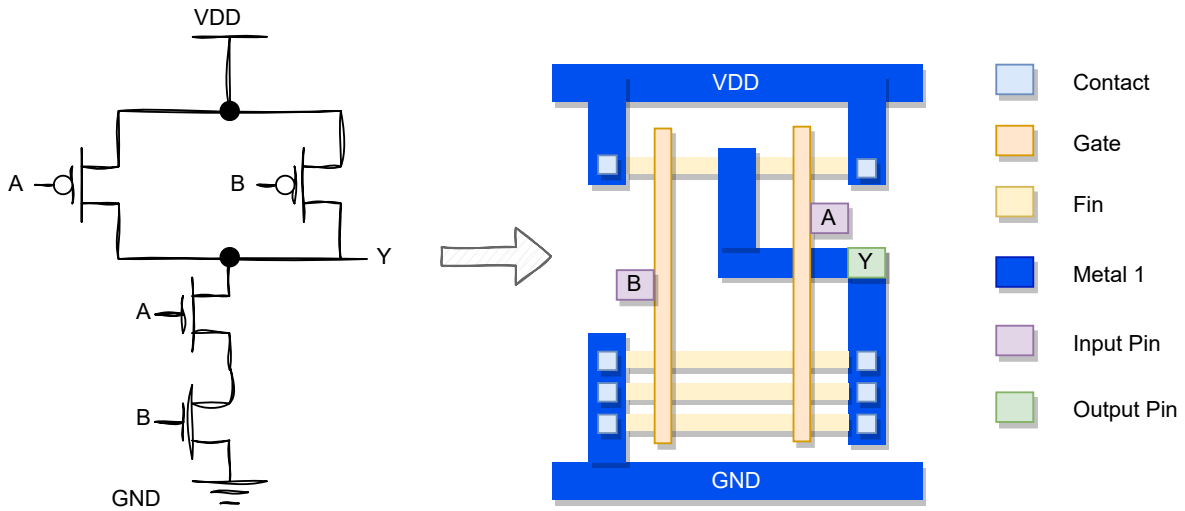
[123] J. He, U. Agarwal, Y. Yang, R. Manohar, and K. Pingali, "Sproute 2.0: A detailed-routability-driven deterministic parallel global router with soft capacity," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, p. 586–591, IEEE, 2022.

[124] T. Cui, J. Li, Y. Wang, S. Nazarian, and M. Pedram, "An exploration of applying gate-length-biasing techniques to deeply-scaled finfets operating in multiple voltage regimes," *IEEE Transactions on Emerging Topics in Computing*, vol. 6, no. 2, pp. 172–183, 2018.

Appendix A

NAND2X1 Gate Layout View

A sample 7nm FinFET NAND gate with 2 inputs and 1 output reported by [124] is shown in Figure A.1.



(a) Schematic View of NAND2x1

(b) Layout View of NAND2x1

Figure A.1: A sample standard cell representation of NAND gate with 2 inputs and 1 output reported by [124]

Appendix B

Detailed Router Engines

The results presented in the Table B.1 show that TritonRoute achieves the highest detailed routing score in the majority of benchmarks. In this table, the column score is the detailed routing score. Only in the two small benchmarks, ispd18_test1 and test2, is TritonRoute outperformed by DRCU. Therefore, TritonRoute has been selected as the preferred detailed routing engine.

Table B.1: This table presents the results of the evaluation of three detailed routing engines, LuLuRoute, DRCU, and TritonRoute on the ISPD 2018 benchmarks.

benchmark	Detailed Router	WL	Vias	OFGW	OFGV	OFTW	OFTV	WWW	Short	Min Area	Spacing	Score	Time(s)	Mem (MB)
ispd18_test1	LuLuRoute	185209713	33541	2846370	168	136050	0	4242010	667184800	0	6720	5761603	-	-
ispd18_test1	**DRCU**	**173418520**	**32402**	**871550**	**443**	**165640**	**0**	**2332830**	**16800**	**0**	**2**	**291291**	**8**	**402**
ispd18_test1	Triton	172900705	35447	2767455	747	377855	161	3833985	0	0	0	304904	18.7973	741.13
ispd18_test2	LuLuRoute	3279212514	335990	120537400	3605	1172540	0	30450360	14763411040	0	62507	82542696	-	-
ispd18_test2	**DRCU**	**3133074700**	**325684**	**16534140**	**6542**	**2117070**	**0**	**21654980**	**19600**	**0**	**57**	**4700933**	**81**	**1904**
ispd18_test2	Triton	3146553540	360057	32563850	9044	4519420	2248	32141745	0	0	0	4832011	207.981	1450.5
ispd18_test3	LuLuRoute	3610617115	318133	363487750	6373	821180	0	25544050	44770707540	0	65735	178905476	-	-
ispd18_test3	DRCU	3487122926	318309	29963440	6636	2376740	0	24183400	35599800	0	94	5298745	120	2145
ispd18_test3	**Triton**	**3503841165**	**355898**	**36608885**	**8450**	**4874455**	**2318**	**33052780**	**2112000**	**0**	**21**	**5299713**	**373.854**	**1698.67**
ispd18_test4	LuLuRoute	5543018572	632047	668496719	21899	761104	0	54293378	52663697820	13264	99573	733474118	-	-
ispd18_test4	DRCU	5284058129	729312	75627486	29895	3488953	0	41042655	9813340	93	593	15756404	720	9744
ispd18_test4	**Triton**	**5246843336**	**725626**	**42093712**	**40500**	**10917940**	**17343**	**32411419**	**2036000**	**0**	**46**	**15074474**	**431.675**	**2832.46**
ispd18_test5	LuLuRoute	5837060163	865651	1100323413	38384	1939580	0	9342780	49746732000	26694	156926	735559666	-	-
ispd18_test5	DRCU	5560260092	965544	30030150	21868	1165333	3	15468553	2498140	141	362	16366742	417	7296
ispd18_test5	**Triton**	**5533810430**	**892269**	**59258695**	**26826**	**4932380**	**16028**	**21484260**	**0**	**0**	**0**	**16077964**	**378.238**	**3049.08**
ispd18_test6	LuLuRoute	7500990737	1415901	252278886	14228	2394100	0	30268980	14188796140	80081	223799	352317181	-	-
ispd18_test6	DRCU	7140701827	1480617	48621481	36581	3296581	16	23802281	576000	265	542	21630645	510	6723
ispd18_test6	**Triton**	**7112168600**	**1374103**	**111742895**	**47525**	**10430640**	**24758**	**33451300**	**0**	**0**	**0**	**21352958**	**587.345**	**3997.25**
ispd18_test7	LuLuRoute	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed	Failed
ispd18_test7	DRCU	13034682078	2402543	80710061	55532	6512839	0	37293214	6565568	357	196	38412191	1028	12827
ispd18_test7	**Triton**	**12964945480**	**2237266**	**171266430**	**73044**	**17629580**	**28548**	**51899750**	**0**	**0**	**0**	**38148393**	**1194.05**	**5349.34**
ispd18_test8	LuLuRoute	13911876354	2353764	201249466	82478	75047174	22294	91164831	3071583160	48	161229	161426598	-	-
ispd18_test8	DRCU	13093815213	2412121	80857765	54369	6478835	0	36526384	7175536	401	192	38602461	1150	12884
ispd18_test8	**Triton**	**13022372820**	**2253510**	**167997530**	**75446**	**17439540**	**28544**	**51407100**	**0**	**0**	**0**	**38307564**	**1195.89**	**5313.94**
ispd18_test9	LuLuRoute	11484579286	2306399	434022520	19254	3015840	0	41669920	25072710120	124830	383340	603223377	-	-
ispd18_test9	DRCU	10952057744	2410790	71698090	55318	5271443	0	35517559	593200	522	109	33129214	814	11457
ispd18_test9	**Triton**	**10882227420**	**2246531**	**167352000**	**73219**	**16196720**	**28786**	**49429600**	**0**	**0**	**0**	**32925035**	**1045.68**	**5280.8**
ispd18_test10	LuLuRoute	13548295780	2439382	934634001	27945	3524440	0	41136800	53462778220	128049	392663	973977612	-	-
ispd18_test10	DRCU	13618037296	2594386	213340113	97600	7971242	0	47221944	103283900	600	918	42704252	1814	11891
ispd18_test10	**Triton**	**13535275510**	**2428654**	**239568520**	**114008**	**21359020**	**32126**	**60713100**	**216000**	**3**	**0**	**40400636**	**1698.79**	**5451.5**

Appendix C

ISPD Benchmarks Detail

In the following, detailed information about the design of each circuit in the ISPD benchmarks is given.

- ISPD18_test1 (Figure 5.1 (A)): Standard cells without IO pins.
- ISPD18_test2 (Figure 5.1 (B)): Standard cells with IO pins.
- ISPD18_test3 (Figure 5.1 (C)): Standard cells with IO pins and macro blockages.
- ISPD18_test4 (Figure 5.1 (D)): Standard cells with metal2 obstacles and different sizes of macro blockages.
- ISPD18_test5 (Figure 5.1 (E)): Metal2 obstacles, with reversed routing direction and boundary macro blockages.
- ISPD18_test6 (Figure 5.1 (F)): Metal2 obstacles, Metal2 power/ground pins, and reversed routing direction.
- ISPD18_test7 (Figure 5.1 (G)): Quad-core design with Metal2 obstacles, and Metal3 power/ground as blockages.
- ISPD18_test8 (Figure 5.1 (H)): Quad-core design with Metal2 and Metal3 obstacles, and Metal2 and Metal4 power/ground as pins blockages.
- ISPD18_test9 (Figure 5.1 (I)): Quad-core design with Metal2 to Metal3 obstacles and Metal2 to Metal4 power/ground pins as blockages. This circuit has no macro blockage, however, it has higher cells' utilization compared with ISPD18 test6 and test7. The cell utilization may cause several issues such as routability and the complex legalization process to find a no-overlap solution, and it is also hard to improve the quality of placement due to dense local areas.

- ISPD18_test10 (Figure 5.1 (J)): Quad-core design with Metal2 to Metal3 obstacles and Metal2 to Metal4 power/ground pins as blockages. There are no macro blockages in this circuit, however, there is 100% cell utilization in the pin layer. Note: The 100% cell utilization in the first layers is modeled by extra cell blockages called Filler in this circuit.
- ISPD19_test1 (Figure 5.2 (A)): Single-core design with complex pin access shapes and special nets.
- ISPD19_test2 (Figure 5.2 (B)): Single-core design with lower boundary macro blockages, and special nets span across the layout with boundary IO pin access congestion. This circuit also has ring VDD/GND special nets in upper layers which makes the routing in the boundary of the layout challenging.
- ISPD19_test3 (Figure 5.2 (C)): Dual Tone Multi-Frequency (DTMF) design with 3 memory blocks and Phase-Locked Loop (PLL) blocks. In this design, the inner macro blockages impose complex challenges in routing such as narrow channel routing and highly congested routing areas.
- ISPD19_test4 (Figure 5.2 (D)): Matrix multiplier design with 7 inside core macro blockages, with poor initial placement, and restricted number of routing layers to 5.
- ISPD19_test5 (Figure 5.2 (E)): 32 bit Peripheral Component Interconnect (PCI) bridge design with 6 inside core macro blockages. In this circuit, a poor initial placement with local dense areas and congested routing channels around macro blockages are given.
- ISPD19_test6 (Figure 5.2 (F)): Quad-core design with 16 memory blockages.
- ISPD19_test7 (Figure 5.2 (G)): Quad-core design with double the number of cells 16 memory blockages.
- ISPD19_test8 (Figure 5.2 (H)): Quad-core design with triple the number of cells 16 memory
- ISPD19_test9 (Figure 5.2 (I)): Quad-core design with quadruple the number of cells 16 memory
- ISPD19_test10 (Figure 5.2 (J)): Quad-core design with quadruple the number of cells 16 memory and extra routing blockages.