On the Routability-driven Placement

HE, Xu

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

The Chinese University of Hong Kong
September 2013

Abstract

In traditional VLSI design flow, the processes of placement and routing are done separately. A placer determines the positions of cells and generates a row-based layout. The major objective is to minimize wirelength which is often estimated by the half-perimeter wirelength (HPWL) model. After placement, a router determines the routing path to connect all the pins of the cells in the same signal net. Since the routing capacities of the metal layers are limited, the router not only need to route all signal nets with small wirelength, but also has to satisfy the routing resource constraints. Therefore, although minimizing HPWL in placement can reduce the average routing demand, the wires may be distributed unevenly and overly optimized wirelength may lead to unroutability of some nets. As a result, we need to consider routing congestion during placement. In this thesis, we study how to trade off between routability and wire length in global placement, legalization, detailed placement, and post-placement process.

First, we study the routability problem in global placement. We propose a robust and effective flow to relieve routing congestion. We studied different methods to identify congested regions, including probabilistic approaches and global routing approaches. The accuracy of different methods were also analyzed. After congestion analysis, we will use cell inflation and spreading to reduce congestion. Cell inflation has been a traditional technique to deal with congestion, and we will discuss how this technique can be used easily and robustly.

Second, we study how to optimize wirelength without worsening congestion during legalization and detailed placement. We find that many HPWL-driven approaches will increase congestion and reduce routability. After global placement, the cells are located quite sparsely in the potentially over-congested regions. In

many cases, the cells will be moved closer to each other after HPWL-driven legalization and detailed placement and the overflow will very likely be increased significantly. Unlike many previous works that focus on different types of swapping strategies in detailed placement, we analyze and propose some simple and effective approaches to consider routability. Experimental results show that these approaches can avoid worsening congestion effectively while the reduction in wirelength is not affected obviously.

At last, we propose a fast and effective simultaneous routing and placement refinement tool called Simultaneous Routing and Placement (SRP). SRP is independent of the placer and the global router. Based on a given placement layout and a global routing result, SRP relocates problematic cells by considering routing and placement simultaneously. Not only overflow from local nets, but overflow from global and semi-global nets can also be resolved by SRP. A cell will be relocated and its associated nets will be rerouted if its connections go across congested regions, even if the cell is not located in the congested region. Experimental results show that our method can reduce overflow effectively. Given the layouts generated by the top four routability-driven placers in the DAC Contest 2012, SRP can still reduce the total overflow by 32.6% on average while the routed wirelength and HPWL are not increased obviously.

Our placer Ripple combines the above approaches in global placement, legalization, detailed placement, and post-placement process. As far as we know, comparing with other placers on the latest benchmarks released in the ICCAD 2012 contest, the routability of our placement result is the best on average.

摘要

在超大規模集成電路物理設計中，布局和布線一般是分開進行的。布局階段計算電路單元在芯片中的位置，生成基於行的布局結果。布局階段的主要目標是優化線長，但因為布局階段很難計算準確的線長，所以是用線網矩形框的半周長模型 (HPWL) 來估計線長。得到布局結果後，布線器會決定每個線網該如何走線，使得同一線網的所有單元連在一起。因為芯片上每個區域的布線資源是有限的，所以布線器不僅需要減小線長，而且每個區域的走線條數不能超過布線資源。雖然布局階段優化HPWL能減少總的布線長度，但是走線通常分布不均勻。如果局部出現布線擁擠，那麼一些線網就會布不通。目前，考慮布線擁擠的布局算法已經成為工業界和學術界討論的熱點問題。本論文主要研究如何在布局的每個階段，包括整體布局、合法化、詳細布局、後處理等，同時優化線長和可布線性。

首先，我們研究在整體布局階段如何考慮可布線性。我們提出了一套有效的流程來優化布線擁擠問題。對中間布局結果，我們嘗試了各種不同的方法對芯片的各個區域估算布線擁擠度，包括概率方法和總體布線算法。並且，我們對不同估算方法的準確度也進行了分析。在估算完布線擁擠度之後，我們膨脹電路單元面積並擴散這些電路單元，使得在布線擁擠度高的區域，電路單元盡量分布稀疏。采用電路單元膨脹擴散的方法來降低布線擁擠度是個很常見的技術，但是如果過度膨脹反而會同時增加線長和擁擠度。所以，我們對如何簡單有效地使用這個技術，進行了深入的分析。

其次，在布局合法化和詳細布局階段，我們研究如何在優化線長的同時，不增加布線擁擠度。傳統的布線驅動算法會影響布線擁擠度，從而降低可布線性。一般在整體布局結果中，在可能的布線擁擠區域，電路單元會分布得很稀疏。但是，如果合法化和詳細布局階段僅僅考慮線長，這些電路單元很可能被重新放置得很緊密，從而降低可布線性。很多考慮可布

線性的算法，基本是關於如何選擇電路單元進行交換的局部策略。不同於這些算法，我們通過實驗，分析並提出了一些簡單有效的方法來解決可布線性問題。實驗表明，這些方法能有效的防止可布線性降低，同時相比傳統的布線驅動算法，線長也沒有明顯增加。

最後，我們提出同時進行布局和布線優化的後處理工具SRP。SRP獨立於布局和布線工具。輸入布局結果和相應的總體布線結果，SRP會對造成布線擁擠的電路單元重新布局布線。由於重布局布線不僅針對在擁擠區域的單元，而且也針對走線經過擁擠區域的單元，所以不僅能優化局部布線擁擠，而且也能優化由全局走線造成的擁擠。實驗表明，我們的方法能有效減少布線擁擠度。給定DAC 2012競賽前四名的布局結果，SRP可以平均優化布線擁擠度32.6%，同時HPWL和實際走線長度都沒有明顯增加。

把布局的各個階段整合在一起，就是我們的布局工具Ripple。Ripple獲得ISPD 2011布線驅動的布局競賽冠軍，以及DAC 2012, ICCAD 2012競賽亞軍。目前，使用最新由ICCAD 2012競賽提供的布局測試用例，同其他布局工具相比，Ripple的平均可布線性最好。

Acknowledgments

First, my greatest thanks and appreciation go to my supervisor Evangeline F. Y. Young. Without her insightful guidance, advice, and continuous Encouragement, this thesis would not have been possible. I have learned a lot from her kindness, patience, and positive attitude towards life. All these would be invaluable throughout my life. I'm especially grateful for the days back to 2010 when she introduced the routability-drive placement problem to me. Little did I know that this elegant problem would bring me so much fun, hard work, frustration, satisfaction, and fulfillment.

Also I would like to thank fellow lab mates Huang Tao, Xiao Linfu, Jiang Yan, Tian Haitong, Cui Guxin, Qian Fuqiang, Chow Wing Kai, Kuang Jiang, and Cai Wenzan. Thanks for all your helps during my Ph.D. study. I won't forget the meetings we discussed together for the contests. I would also like to thank my friends Wei Xin, and Diao Yi, Chen Zhitang, Xu Jinxi and et al for sharing happiness during the last three years.

Finally, I want to express my heartfelt thanks to my parents Li Yueyue and He Dongliang, and my relatives Wen chunling, Li Yunliang, Wang Shifen, He Huilan, Li qingqing, Li bingbing and et al for their love, understanding, and constant support. Their encouragement has always been a powerful source of inspiration and energy. Without them, this dissertation would not exist.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Contents

1.1 Introduction of VLSI flow

Because of the complexity of Very Large Scale Integration (VLSI), there are several steps in the VLSI design cycle as shown in Fig. 1.1.

Given a system specification which is a high level description of the system, the step of architectural design decides the architecture of the chip, e.g., RISC versus CISC, number of ALUs, etc. Then, the step of functional design identifies main functional units of the system and interconnect requirement between them.

It also estimates the area, power and other parameters of each unit. Logic design describes the control flow, word widths, arithmetic operations, register allocation and logic operations, which is called the Register Transfer Level (RTL) description. Based on the result of logic design, the task of circuit design is to convert the boolean expression into a circuit representation by considering the speed and power requirement of the system. In physical design, the circuit representation (gates, transistors) is converted into a geometric representation with specific shapes on multiple layers. Based on the circuit layout obtained by physical design, the chip is ready for fabrication on a wafer. At last, each individual chips will be packaged and tested to ensure whether it meets all the design specifications and functions properly.

1.1.1 Physical design

In the process of VLSI design, physical design is the most important step. The objective of physical design is to give optimal arrangements of millions of devices on a plane (or a number of planes) and determine an effective interconnection schemes between the devices. Currently, there are usually millions of transistors on a chip. Due to the large number of devices, physical design must resort to the support of computers.

Fig. 1.2 shows the process of physical design. The input is a circuit diagram; while the output is the layout of the circuit. There are several phases in the flow of physical design:

1. Partitioning: Because of the huge number of transistors on a chip, it is impossible to layout the whole circuit in one step. Usually, a circuit is partitioned into many sub-circuits in a preprocessing step. These sub-circuits are called blocks. The output of this partitioning step is a set of blocks and interconnections between

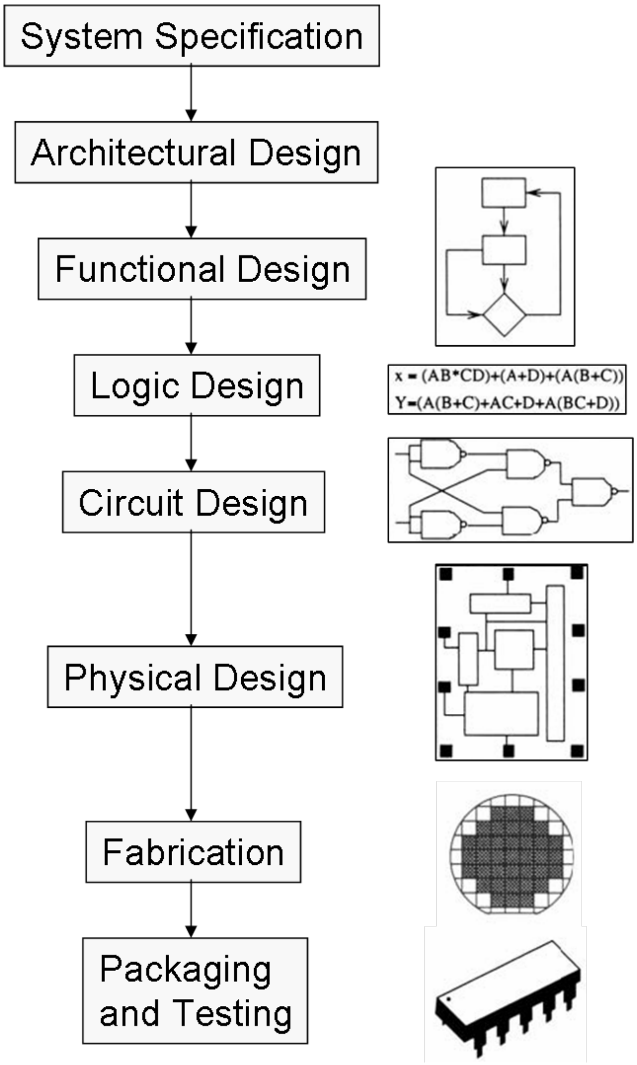


Figure 1.1: VLSI design flow [47]

them which are called netlist information.

2. Floorplanning and Placement: The positions of blocks are determined in this step. Floorplanning can provide a guideline for locating the functional blocks. The number of the functional blocks in floorplanning is usually several hundreds. However, placement has to place millions of small cells and most of these cells are rectangular shape with fixed width and height.

3. Routing: The interconnections between blocks according to the netlist are

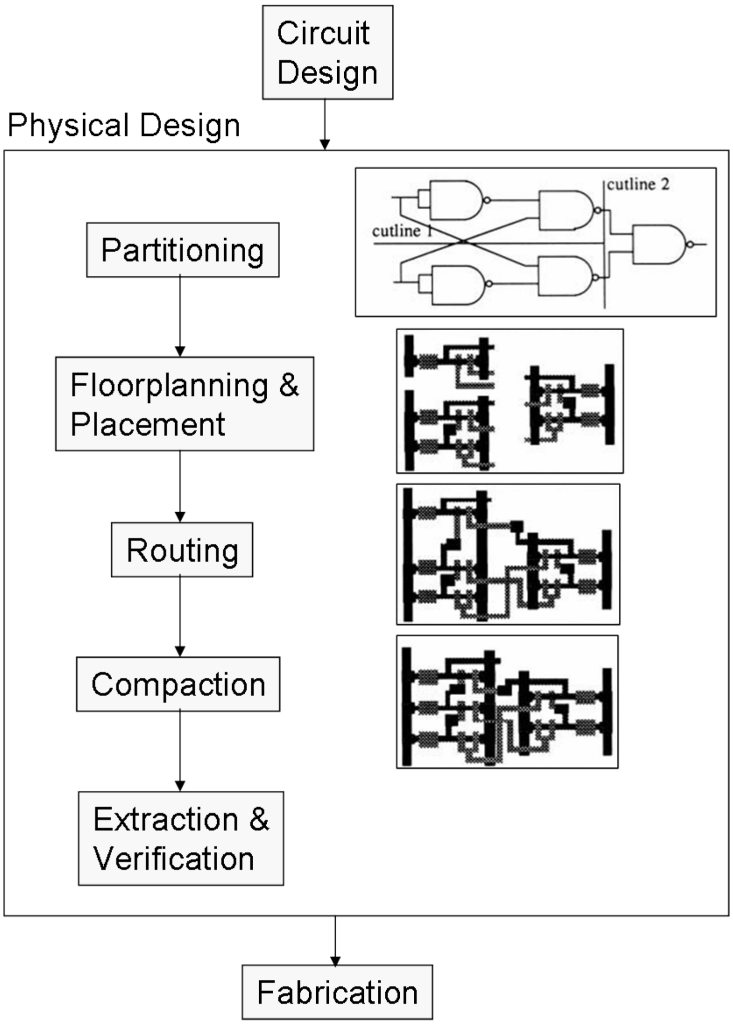


Figure 1.2: Physical design flow [47]

completed in this step. Nowadays, there are several metal layers above the device layer for routing, and over-the-device routing has become popular. For each metal layer, its wire direction is usually either horizonal or vertical. The route from the source of a net may go through several layers until reaching the sink of the net. Because of the complexity of the routing problem, it is solved in two phases: global routing and detailed routing. In global routing, the routing area is represented as a grid graph, and the interconnections across grids will be roughly routed. Based on the global routes, detailed routing will be done to find the exact routes of all the nets.

4. Verification: After getting the layout, Design Rule Checking (DRC) will be performed to verify that all geometric patterns meet the design rules required by the fabrication process. Besides, the functionality of the layout needs to be verified in this step as well.

1.2 Placement problem formulation

In the placement phase, a block/gate/ transistor-level netlist is transformed into an actual layout for implementation. Because the locations of the circuit elements are obtained during placement, the corresponding route and delay of the interconnection will be affected greatly. Therefore, the placement result has significant impact on the final performance of the design and is regarded as one of the most important optimization step in physical design.

1.2.1 Traditional placement

Since the circuit elements have to be located on a chip, we should define the placement region P first. The region P is usually a rectangle with coordinates (x_{low}, y_{low}) and (x_{high}, y_{high}). The circuit netlist is represented as a graph $G = (V, E)$, where V is a set of circuit elements and E is a set of interconnections (nets) between the circuit elements.

The vertex set V includes two disjoint subsets: MV and FV, where MV and FV represents a set of movable and fixed circuit elements respectively. The location of each circuit element $v \in FV$ has already been determined and cannot be moved. Therefore, the task of placement is to give a location (x, y) to each circuit element $v \in MV$ such that the total wirelength is minimized. Other objectives such as routability, noise, temperature will also be considered in some placers.

Each net $e \in E$ connects a subset of circuit elements, i.e., $e = \{v_1, v_2, \ldots, v_m\}$, where $v_i \in MV \cup MF$. There is a pin contained in each circuit element v_i for this connection, and its position in v_i is fixed.

Suppose that in $G = (V, E)$, there are n circuit elements $V = \{v_1, v_2, \ldots, v_n\}$ and m nets $E = \{e_1, e_2, \ldots, e_m\}$, i.e., $|V| = n$ and $|E| = m$. For each movable circuit element $v_i \in MV$, its location is a variable (x_i, y_i). The placement problem can be defined as follows: Given a placement region P with width W and height H, a netlist $G = (V, E)$ and an objective function $f(V, E)$ which is typically formulated as the total wirelength of all the nets, the task is to find the location (x_i, y_i) for each movable circuit element $v_i \in MV$, such that v_i is:

- Located within P without overlapping with other circuit elements;
- The objective function $f = (V, E)$ is minimized;
- In the case of the standard cell placement problem, additional circuit rows are designated in the region P, and the circuit elements, which are called standard cells, with the same height must be placed in the rows.

Currently, hierarchical design methodology is very popular to reduce the turn-around-time of taping out chips. Because of this, more and more macro blocks (either fixed or movable) exists in modern ASIC designs, e.g.. reusable internal/third-party IPs. Therefore, the circuit elements in V can have a large range of sizes, and this type of placement problem is called mixed-size placement. In mixed-size placement, besides those small standard cells, there are some large macro blocks. Fig. 1.3 shows two examples of pure standard cell placement and mixed-size placement.

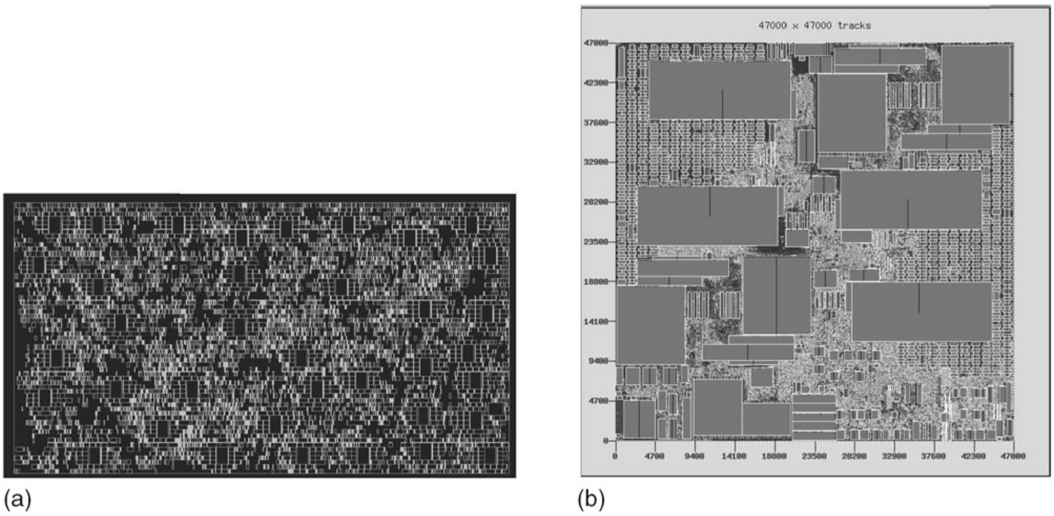


Figure 1.3: (a) Pure standard cell placement (b) Mixed-size placement [3].

In earlier VLSI design flow, placement and routing were implemented in separate software tools. A placer generates row and site-aligned, non-overlapping locations for the circuit elements with small estimated interconnect length. After placement, a router determines the routing path to connect all the pins of the circuit elements in the same signal net.

The traditional placement objective is usually measured by the estimated wirelength, like the half-perimeter bounding box wirelength model (HPWL), rectilinear steiner tree wirelength model, and so on. Fig. 1.4 shows the results of these estimation models on a three-pin net.

HPWL is widely used when comparing results in placement. HPWL is easy to measure and yet a reasonable first-order estimation for the routed wirelength of small nets in particular. For larger nets, steiner tree wirelength is needed to accurately estimate the routing wirelength [38].

However, the traditional placement metric is hard to capture the key aspect of solution quality. Since the routing capacity of metal layer is limited, a router not only needs to route all signal nets with small wirelength, but must also satisfy the

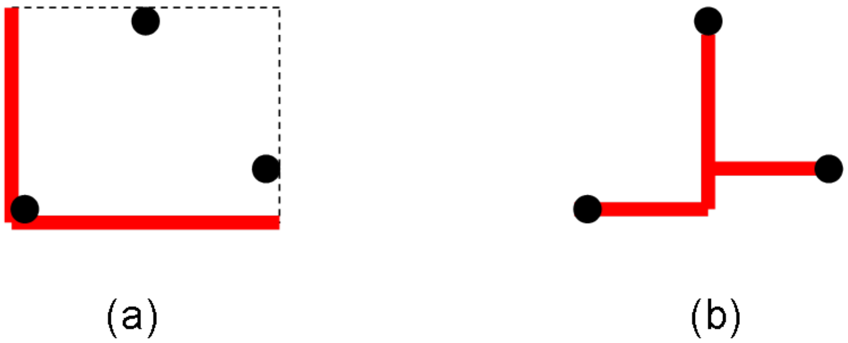


Figure 1.4: (a) HPWL model on a three-pin net (b)Rectangular steiner tree model on a three-pin net. The estimated wirelength is the length of bold lines in red color.

routing resource constraints. Therefore, although minimizing HPWL in placement can reduce the average routing demand, the wires may be distributed unevenly and overly optimized wirelength may lead to unroutability of some nets.

1.2.2 Routability-driven placement

If a placement solution is not routable in the following routing phase, it must be re-designed again. Placement solution has to be routable. Currently, one of the key challenges for modern physical synthesis flow is routability [2]. There are several factors that contribute to the issue of routing congestion in advanced process technologies like 65nm and below. A few of them being, increased use of embedded IPs or memories on the die that blocks metal layers, more layer stacks to achieve higher performance, reduced die size to control manufacturing cost, and complicated logic structures such as cross-bars. As a result, physical synthesis needs to consider routing congestion in the entire design flow [54].

To address this issue of routing congestion, routability-drive placement becomes a hot topic in recent years and several routability-driven placement contests were held in some top conferences like ISPD 2011 [54], DAC 2012 [52] and ICCAD 2012 [55] and attracted many researchers. The routability problem and the evaluation metric are formulated in these contests. Besides giving the information of the placement region P and the circuit netlist $G(V,E)$, the routing information is also provided. Typically, a global router will overlay a regular grid (GCells) on the chip, and construct a global routing graph (Fig. 1.5 (a)). Each node in the graph represents a GCell in the layout, and an edge (GEdge) represents the boundary between adjacent GCells. In modern circuits, there are more than one metal layer for routing, (e.g., nine layers in modern designs). The 2D regular grid graphs of each metal layer stack up as a 3D grid graph (shown in Fig. 1.5 (b)). Via edge is used to connect two abutting GCells in adjacent layers. To enable global routing and congestion analysis, the routing information is given as follows:

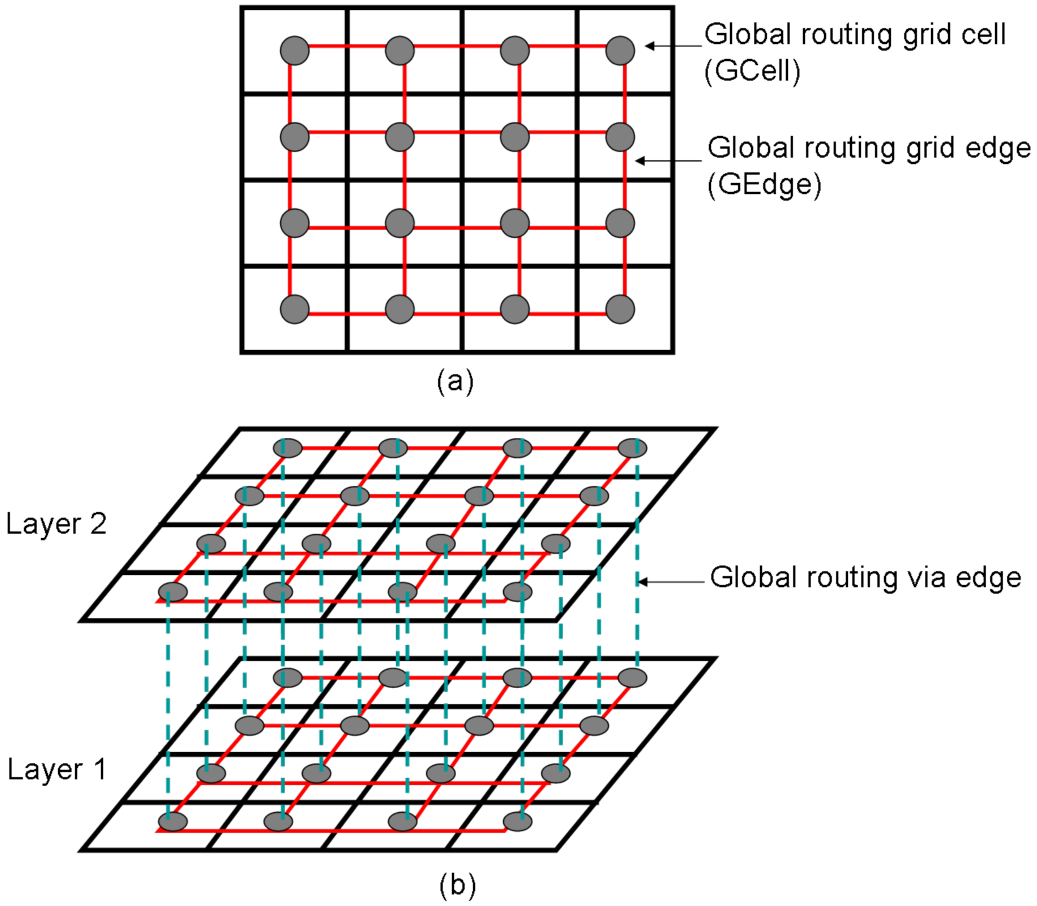

Global routing grid cell (GCell)
Global routing grid edge (GEdge)
(a)
Layer 2
Global routing via edge
Layer 1
(b)

Figure 1.5: (a) 2D global routing graph for one metal layer (b) 3D routing graph.

- Global routing grid (grid size, GCell dimensions, etc.)
- Number of routing layers
- Maximal GEdge routing resource for each routing layer
- Wire width and spacing for each routing layer
- Via specification
- Routing blockage information
- Routing layer for pins of random logic macros (RLM)

According to these routing information, the width or height of each GCell and capacity of each GEdge can be obtained. The capacity $c(e)$ of an GEdge e is the number of routing tracks that can legally go across the adjacent grids, and demand $d(e)$ of e is the number of global rotuing paths passing through e. The overflow of e is defined as $max(0, d(e) - c(e))$. If $d(e) > c(e)$, it implies that too many global routing pahts are passing through e, resulting in routing overflow.

Different from the contests held in ISPD 2005 [37] and ISPD 2006 [36] which primarily evaluate a placer based on the HPWL metric and use density target to address routability and congestion mitigation [38], a routability-driven placement solution is evaluated by some metrics defined based on a global routing result.

In the ISPD 2011 contest, a global router coalesGrip [48] is used to route each placement solution within a time budget (e.g., 15 min). The total overflow of all the GEdges reported by the global router is used to evaluate the routablity of a placement solution, i.e., lower total overflow implies better routability. However, the metric of total overflow usually fails to provide a clear picture of the design routability [60]. Considering only total overflow may encourage placers to use routings with very long detour to reduce a small amount of overflow, and it may add to the difficulty in detailed routing. In addition, as the peak overflow is not considered, a routing solution may have very congested hot-spots although the total overfloap overflow may not be alarming. As shown in Fig. 1.6, the total overflow value of SimPLR [29] is less than that of Ripple [18]. However, the distribution of the congestion hotspots in SimPLR is more uneven, and it has worse routability when compared to Ripple on this design [52].

In order to overcome the limitation of the total overflow metric, a new congestion metric called average congestion of GEdges (ACE) to give a more accurate congestion measure. $ACE(x)$ computes the average congestion in percentage of the top x% congested GEdges. Using different values of x, e.g. $x \in \{0.5, 1, 2, 5, 10\}$, a set of ACE values can be calculated, which together can provide a more accurate view of the design congestion. For example, $ACE(0.5)$ or $ACE(1)$ provide a highly local view, representing the most congested regions in the design. On the other hand, $ACE(5)$ or $ACE(10)$ give a broader view of the design congestion [52]. Let,

- Peak Weighted Congestion (PWC):

$$PWC = \frac{\sum_{x \in \{0.5,1,2,5\}} K_x \times ACE(x)}{\sum_{x \in \{0.5,1,2,5\}} K_x}, \tag{1.1}$$

  where K_x is the weight of $ACE(x)$.

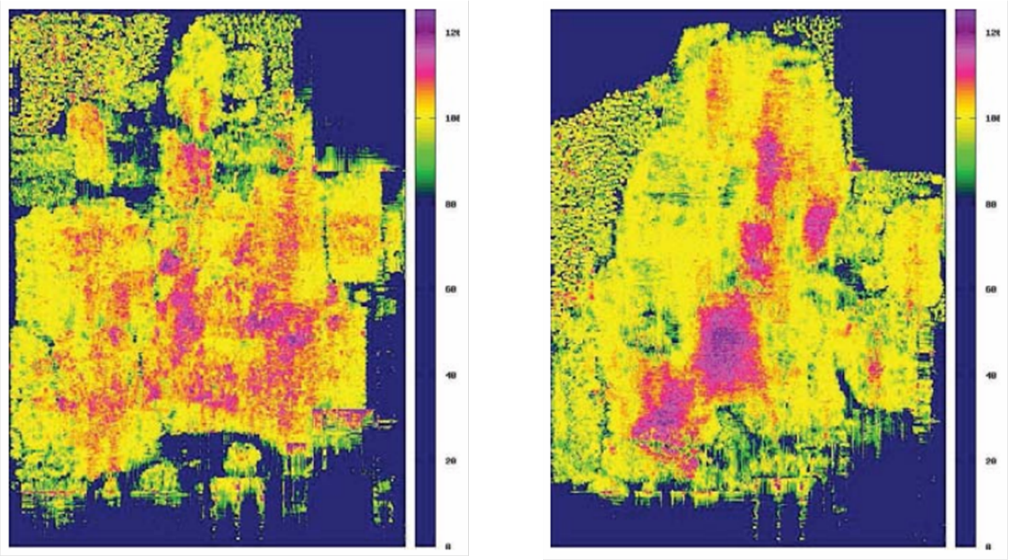
- Routing Congestion (RC):

$$RC = \max(100, PWC) \tag{1.2}$$

By considering the HPWL and the routing congestion (RC), the routability evaluation metric of a placement solutioncan be modeled by Equation (1.3).

$$scaledWL = HPWL \times (1 + PF \times (RC - 100)) \tag{1.3}$$

where, PF is a penalty factor that scales the HPWL to account for over-congestion.



Ripple with total overflow of 542786 SimPLR with total overflow of 514614

Figure 1.6: Congestion maps and total overflow values for Ripple [18] and SimPLR [29] placement solutions on superblue12. The regions colored purple indicate congestion hot-spots [52].

Local peaks of pin density within GCells often cause internal routing congestion, but are usually overlooked by global routing that only captures the wires passing through GEdges. In the metric of ICCAD 2012 [55], congestion due to local wires (intra-GCell routes) are considered. Based on the metric of ICCAD 2012, when computing ACE values, the routing demand of a GEdge e equals the sum of the routing tracks passing through e and a weighted number of pins within the two adjacent GCells connected by e.

1.3 Procedural flow in placement

Placement is a complicated NP-complete problem. A placement problem is usually solved by four steps: global placement, legalization, detailed placement, and postprocess. Global placement determines the approximate distributions of the circuit elements to optimize the objective function. In global placement, the circuit elements may have some degree of overlapping. Then, the legalization step will transform an illegal global placement result into a legal one (i.e., remove overlaps, and locate cells within rows in the case of standard cell placement). The detailed placement step is performed to improve the objective function further. Usually, the legalization step is viewed as a part of the detailed placement process. Fig. 1.7 shows the layout before and after legalization in the case of standard cell placement. In post-process, various refinement methods can be applied for different objective, e.g., power reduction, timing optimization, and routability alleviation, etc.

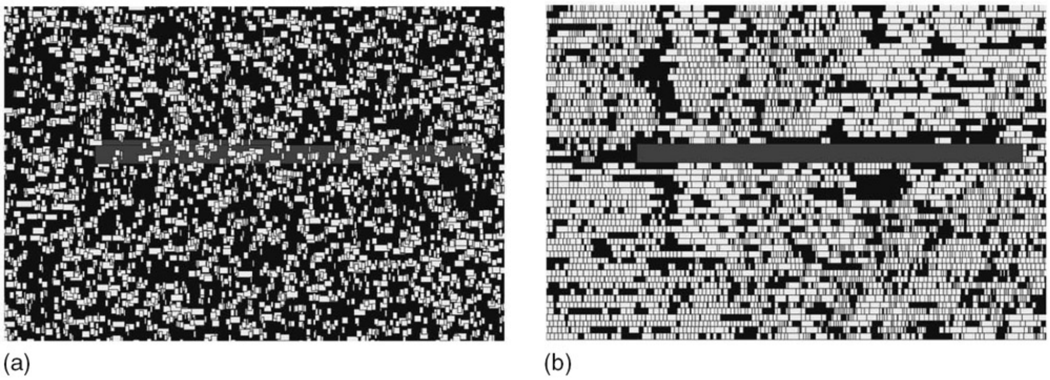


Figure 1.7: (a) Before legalization (b) After legalization [3].

1.4 Thesis outline

This dissertation studies the routability problem in placement.

In Chapter 2, we do a literature review of the placement problem, including the traditional HPWL-driven placement and routabilty-driven placement. We introduce a set of heuristics algorithms used in the state-of-the-art placers for both the HPWL and routability optimization.

In Chapter 3, we propose several techniques in global placement to improve routability. Our global placement is a flat placer with a lower-upper-bound framework [30]. In the lower bound computation, we focus on minimizing HPWL. In the upper bound computation, we spread cells by considering routing congestion. In order to identify congested regions during the upper bound computation phase, we tried many methods, such as probabilistic approaches and global routing techniques. The accuracy of these methods will be discussed in detail. After congestion analysis, we mainly use cell inflation and spreading to reduce congestion. Although this technique has been used in many previous works, it is not easy to be applied effectively. In this chapter, we studied three important questions. Where should cell inflation be used? What should be the inflation ratio? How should the cells be spread? Our algorithm can compute and adjust the cell inflation ratio efficiently without causing over-inflation. Besides, both congestion from local nets and global nets are handled. We also give an effective flow to reduce congestion without scarifying wirelength unnecessarily.

In Chapter 4, we study the routability-driven legalization and detailed placement. Traditional HPWL-driven legalization and detailed placement may increase congestion and reduce routability. We find that after global placement, the cells are located quite sparsely in some potentially over-congested regions. However, after legalization and detailed placement, the cells are moved even closer to each other and the overflow will very likely be worsened significantly. In this chapter, we study the impact of each step of legalization and detailed placement on congestion, and propose some simple and effective methods to trade off between congestion and HPWL in those steps.

In Chapter 5, we propose a fast and effective simultaneous routing and placement refinement method called SRP. SRP is independent of the placer and global router being used. Based on a given placement layout and a global routing result, three major steps are employed in SRP to do congestion refinement. First, we identify and rip-up problematic cells. Different from many previous works that only focus on reducing the overflow caused by local nets, our method may relocate a cell even when it is not lying inside a congestion region but its routing path has run across congested regions. Second, we will search for a new location for the ripped-up cell by a multi-source propagation step. Unlike other works that only search the surrounding region around the cell's original location, we will find a new location for the problematic cell by searching around the G-Cells passed through by the routing paths of the associated nets of the cell. Third, we will connect the problematic cell from its new location to its associated nets by a multi-subnet maze routing algorithm.

In Chapter refchap:result, the experimental results are analyzed. In Chapter 7, a conclusion of this thesis is drawn.

1.5 Thesis contributions

The contributions of this dissertation can be summarized as follows.

For routability problem in global placement:

- Devise an efficient and robust flow of global placement to gradually improve both the congestion and HPWL. Even with accurate congestion estimation, the problem of how and when to utilize the congestion information to improve both the congestion and HPWL is still an open and non-trivial problem.

- Propose a routing path-based method for cell inflation and spreading. Different from previous methods that only spread cells in congested regions, we also consider cells whose routing path passes through congested regions. Our strategy can thus alleviate both local and global congestion effectively.

- Present an simple and stable method to calculate the inflation ratio for cells in the congested regions. Our inflation method can compute the inflation ratio robustly without the need to set any sensitive parameters manually. Besides, for those benchmarks whose congestion measures are too large, our method can adjust the inflation ratio dynamically to avoid over-inflation.

- Present a method of handling tangled logic (a group of cells with many interconnections) to trade off between congestion and HPWL. We detect the congestion due to tangled logic, and spread the cells there more sparsely without sacrificing the HPWL unnecessarily.

For routability problem in detailed placement:

- Legalization is usually thought to have less impact on the placement result since we just legalize the global placement solution. However, we find that this is not true and legalization can worsen congestion in the horizontal direction quite significantly. We analyze the problem, and propose a very effective method to solve it.

- Traditional HPWL-driven detailed placement may worsen congestion a lot. Many previous works may use complex methods to deal with it, e.g., congestion-driven cell swapping. We have studied carefully in various detailed placement steps to identify the problems and developed some simple and effective solutions to handle them.

For routability refinement:

- We present a routability refinement method called SRP that can be used independent of any placer and router. Other refinement tools can be integrated with our tool for further improvement.

- We relocate problematic cells by considering routing and placement resources simultaneously. Not only overflow from local nets, but overflow from global and semi-global nets can be relieved. We relocate and reroute a cell if its connections run across congested regions, even if the cell is not lying in a congested region itself. Results show that our method can reduce overflow effectively.

Finally, an integration of the above ideas and techniques composes a powerful placer called Ripple for routability optimization. In recent years, routability-drive placement contests have been held in ISPD 2011 [54], DAC 2012 [52] and ICCAD 2012 [55] respectively. Ripple won champion once and first runner-up twice in these contests. As far as we know, comparing with other placers on the latest benchmark released in the ICCAD 2012 contest, the routability of our placement results is the best on average.

CHAPTER 2

Background

Contents

2.1 Traditional placement

Typically, the objective of placement is to minimize the total wirelength of all the nets. Smaller wirelength always consume smaller routing resource. Besides, wirelength can also approximate other objectives indirectly, such as timing, power and routability of a design. In addition, wirelength can be modeled relatively easier than other objectives.

Since nets have not been routed yet during placement, we can only estimate the wirelength by using some models and the placement objective is to minimize the estimated wirelength.

Usually, we use the half-perimeter wirelength (HPWL) model. For each net, the HPWL model measures its wirelength as the half-perimeter of the smallest bounding box that surrounds all the pins of the net. Suppose that for a net $e \in E$, the HPWL of net e is:

$$HPWL(e) = HPWL_x(e) + HPWL_y(e) = \max_{v_i, v_j \in e} |x_i - x_j| + \max_{v_i, v_j \in e} |y_i - y_j| \tag{2.1}$$

The total HPWL of all the nets is:

$$TotalHPWL = \sum_{i=1}^{m} HPWL(e_i) = \sum_{i=1}^{m} HPWL_x(e_i) + \sum_{i=1}^{m} HPWL_y(e_i) \tag{2.2}$$

There are some other estimation models, e.g., Rectilinear Steiner Minimum Tree (RSMT), where SMT is a tree that connects all the pins of a net and can make use of any arbitrary point as inserted nodes (steiner point) when building the tree to reduce the tree length. Usually, RSMT is a more accurate estimation of the real wirelength than other models. However, finding RSMT is a NP-complete problem, and it is hard to model this estimation as an analytical function.

The techniques used in traditional placers are summarized in Fig. 2.1. Placers based on stochastic approaches often utilize simulated annealing to find the global optimum layout, but it suffers from long runtime. Partitioning approach is to recursively partition the circuit and the placement area. Analytical approaches optimize an objective function and compute the cell locations by methods of mathematical analysis. We will introduce partitioning and analytical techniques in the following sections.

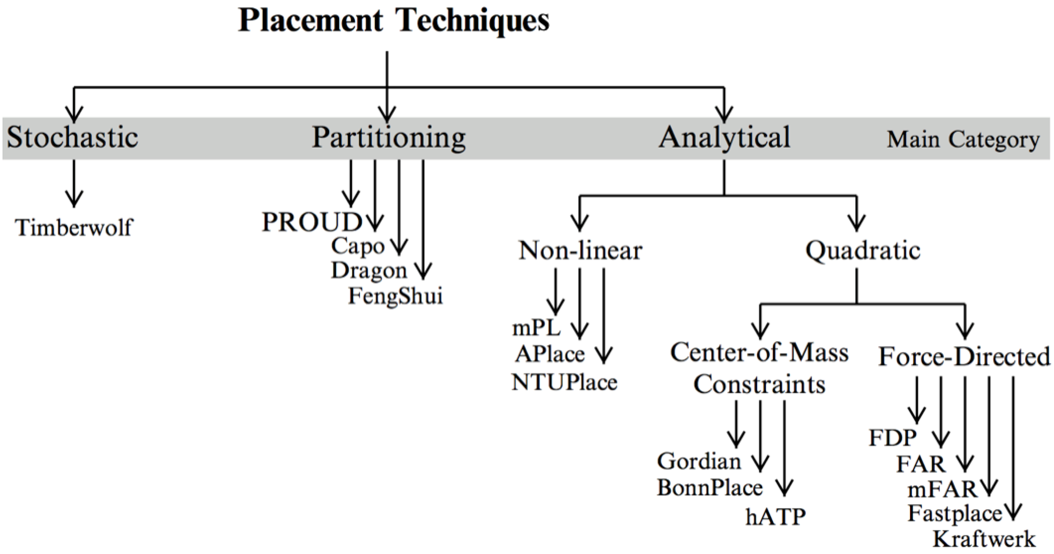


Figure 2.1: Placement techniques and state-of-the-art placers [38].

2.1.1 Partitioning-based placement

Partitioning-based placement has a top-down framework. It recursively decomposes a given circuit into sub-circuits, and divides a placement region into sub-regions (or bins). The cells in sub-circuits are assigned to sub-regions. This algorithm is composed of a sequence of passes where each pass examines all sub-regions and divides some of them into smaller ones. When assigning cells to subregions, we should minimize the number of nets crossing the subregions and consider the total area of cells inside each subregion, which corresponds to a min-cut partitioning problem. Fig. 2.2 (a) gives the process of partitioning-based placement.

2.1.2 Analytical placement

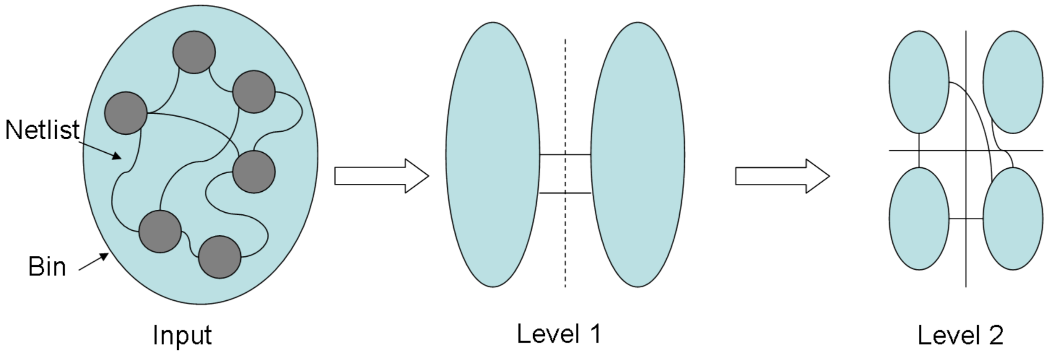
2.1.2.1 Wirelength Approximation

Usually, in analytical placer, the objective function is formulated to minimize the HPWL function shown in equation (2.2). However, equation (2.2) is nonconvex,

```
Variables: A queue Q of placement bins and each bin
           is associated with a sub-cuicuit

Algorithm:
    while(Q is not empty)
        dequeue a bin B
        if(B is small enough)
            Process B with an end-case placer
        else
            Partition the cells in B and put the partitions
          into sub-bins of B
            Enqueue each sub-bins into Q
```

(a)



(b)

Figure 2.2: (a) The high-level outline of the top-down partitioning-based placement process (b) Sequential min-cut partitioning.

and it is hard to be minimized directly [38]. Therefore, some smooth wirelength approximation functions are used:

- Quadratic function:

$$\sum_{e\in E}\Big(\sum_{v_i,v_j\in e,i<j} w_{i,j}(x_i-x_j)^2+\sum_{v_i,v_j\in e,i<j} w_{i,j}(y_i-y_j)^2\Big) \tag{2.3}$$

- L_p-norm:

$$\sum_{e\in E}\Big(\big(\sum_{v_k\in e} x_k^p\big)^{\frac{1}{p}}-\big(\sum_{v_k\in e} x_k^{-p}\big)^{-\frac{1}{p}}+\big(\sum_{v_k\in e} y_k^p\big)^{\frac{1}{p}}-\big(\sum_{v_k\in e} y_k^{-p}\big)^{-\frac{1}{p}}\Big) \tag{2.4}$$

- log-sum-exp:

$$\gamma\sum_{e\in E}(\log\sum_{v_k\in e}\exp(\frac{x_k}{\gamma})+\log\sum_{v_k\in e}\exp(\frac{-x_k}{\gamma})+\log\sum_{v_k\in e}\exp(\frac{y_k}{\gamma})+\log\sum_{v_k\in e}\exp(\frac{-y_k}{\gamma})) \tag{2.5}$$

A placer using equation (2.3) as the objective function is called a quadratic placer; while a placer using equation (2.4) or (2.5) is called a nonlinear-optimization-based placer. For the net models shown in equation (2.3), (2.4) and (2.5), if the problem is convex, it can be solved effectively using Conjugate Gradient method (CG).

Here, we introduce a quadratic net model called Bound2Bound net model as follows. Since the x and y coordinate is independent in Equation (2.3), they can be considered separately. Here, we discuss the objective function in x-direction, and the function in y-direction can be obtained similarly. In the Bound2Bound net model [49], not every pair of two-pin connections of a net is used, but only a few representative ones. Fig. 2.3 shows an example for a net with P pins including pin a and b. Pin a with the smallest x-coordinate is connected to pin b with the largest x-coordinate. We use $l_{x,1}$ to denote the length of this connection. Other $P-2$ inner pins of the net are connected with both outer pins a and b. We use $l_{x,j}$ and $l_{x,j+1}$ where $j = 2,4,...,2(P-2)$ to denote the length of these connections from each inner pin j, and $l_{x,j} + l_{x,j+1} = l_{x,1} = w$, where w is the distance between pin a and b in x-direction. The net creates $1+2(P-2)$ two-pin connections.

The quadratic function of this net is shown in Equation (2.6). This function is equal to the width w of the nets' bounding box in x-direction.

$$\begin{gathered}\sum_{i=1}^{2(P-2)+1} w_i \times l_{x,i}^2,\\ \text{where, } w_i = \frac{1}{(P-1)(l_{x,i})}\end{gathered} \tag{2.6}$$

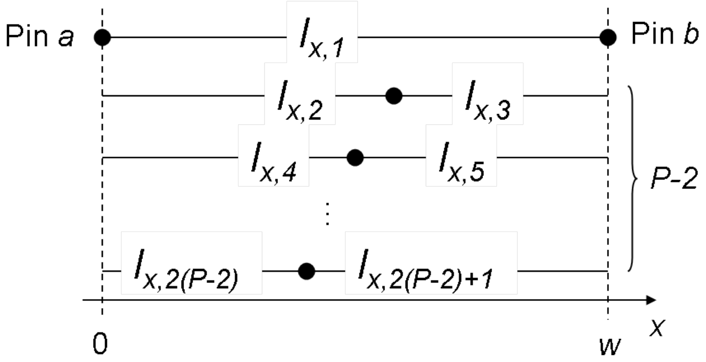


Figure 2.3: The Bound2Bound net model in the x-direction.

Bound2Bound net decomposition capture the HPWL objective exactly, but only for the given placement. When the locations change, the connections and weights of the nets need to be updated throughout the placement algorithm.

After minimizing the objective function and obtaining the location (x_i, y_i) for each movable cell, there will be many overlaps between the cells, and we need other approaches to remove these overlaps, like cell shifting [53], putting cell-density penalty in the objective function [7][27][8], look-ahead legalization [30], and so on.

2.1.2.2 Multilevel framework

When the number of variables is too large, e.g., millions of movable cells will have millions of variables, the runtime for the CG method will be too long especially when the objective function is nonlinear. Therefore, some placers will use a multi-level framework [3] to improve the scalability (shown in Fig. 2.4).

Given a placement problem $G = (V, E)$, we can cluster some cells as a group if there are many connections between them, and this process is called aggregation or coarsening. This coarsening process is performed recursively until the final coarsest-level at which the number of groups is less than a threshold. Then we will use CG to solve the problem and get the positions of the groups. The positions of the groups will be transformed to compute the positions of the subgroups at its adjacent finer level. This process is called interpolation or uncoarsening. This uncoarsening process will be repeated until getting all the original cells.

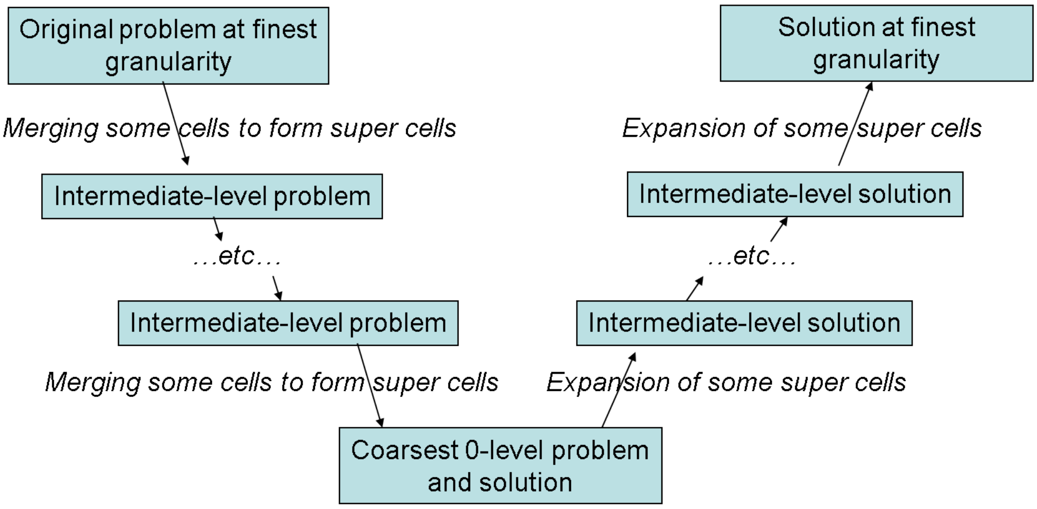

Original problem at finest granularity
Merging some cells to form super cells
Intermediate-level problem
...etc...
Intermediate-level problem
Merging some cells to form super cells
Coarsest 0-level problem and solution
Expansion of some super cells
Intermediate-level solution
...etc...
Intermediate-level solution
Expansion of some super cells
Solution at finest granularity

Figure 2.4: Multilevel framework for placement problem [3].

2.2 Routability-driven placement

In previous placement algorithms, the major objective is to minimize wirelength which is often estimated by HPWL. Although minimizing HPWL can reduce the average routing demand to some extend, the routing demand may be distributed unevenly, and as a result, making some nets difficult to be routed or even unroutable at the end. Therefore, only pursuing short HPWL cannot get routable result with short wirelength as expected, and it may even produce unroutable solution. Moreover, hundreds of large macros that occupy several metal layers usually exist in modern circuit designs. The existence of such routing blockages has made this routability-driven placement problem even more challenging.

Table 2.1: Prior congestion analysis approaches

Category	Specific Techniques
Static	Rent's Rule [65] Net bounding box [5] Steiner tree construction [45] Pin density [4] Nets Counting in bin [58]
Probabilistic	Probabilistic pattern routing [35][61][46] Pseudo-constructive wirelength [28] RUDY (uniform wire density) [50]
Global routing	Using A*-search algorithm on 2D routing grid [62] Integrate FastRoute [63] in placer IPR [42] Using Simplified FastRoute and NCTUgr [34] in placer Ripple [17] Integrate BFG-R [22] in placer SimPLR [29]

There are many previous works on routability-driven placement and we will discuss them in details as follows.

2.2.1 Congestion analysis

In order to alleviate routing congestion, we have to identify the congestion region first. Many prior approaches are proposed for congestion estimation. They can be divided into three main categories: (1) static congestion estimation, (2) probabilistic congestion estimation, and (3) global routing estimation. Table 2.1 [29] gives the summary of these approaches. Traditionally, the first two options are used popularly for their short runtime. Thanks to the improvement in quality and runtime of recent routers, a placement layout can be evaluated by a global router with an acceptable time-out.

Here, we introduce some congestion analysis approaches in detail.

2.2.1.1 Rent's rule

Rent's rule is an empirical observation first descried in [31]. It states the relationship between the number of cells G in a subcircuit of a partitioned design and the number of external connections T of the subcircuit (shown in Equation(2.7)).

$$T = tG^p \tag{2.7}$$

where t is the average number of interconnections per cell and p is the Rent exponent ($0.4 < p < 0.8$ in real circuits). Rent's rule has been widely used to estimate interconnect wirelength in early design [14][51][13].

2.2.1.2 RUDY

RUDY [50] stands for Rectangular Uniform wire DensitY. Since a multi-pin net usually has many different routes if a rectilinear steiner minimal tree (RSMT) is used as a routing model, and all of these routes have the same minimal length, it is hard to predict which routes will be used. Thus, the work [50] suggests that it is not necessary for a single net to predict its routing demand accurately within its minimum bounding box.

RUDY is defined by a uniform wire density d_i for each net $i = 1, 2, \ldots, N$ within the minimum bounding box of net i. The wire density d_i of net i is the ratio of the wire area wa_i and the net area na_i (shown in Equation (2.8)).

$$d_i = \frac{wa_i}{na_i} \tag{2.8}$$

The net area na_i is equal to $w_i \times h_i$, where w_i and h_i are the width and height of the minimum bounding box of net i. The wire area wa_i is computed by the wirelength

l_i times the wire width p: $wa_i = l_i \times p$. The wire width p is defined by the average wire-to-wire pitch divide by the number of routing layer. The wirelength l_i is the estimated routed wirelength of net i and can be computed by the HPWL or by the length of the RSMT. RUDY uses the HPWL as the estimation for the routed wirelength.

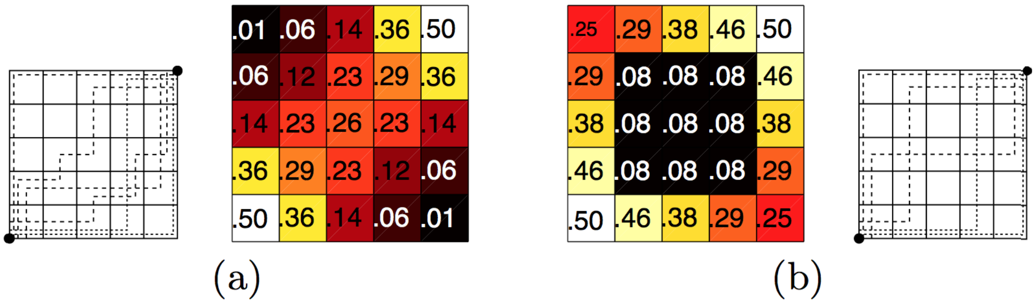
2.2.1.3 Probabilistic pattern routing

The chip is divided into uniform grids as the global router does. For each net, there are a number of possible routing paths, and the probabilistic usages are assigned to the grids that the paths go through. Usually, the analysis is based on two-pin nets, and multi-pin nets are decomposed into two-pin nets (e.g., using a Minimum Spanning Tree algorithm).

For each two-pin net, Lou's model [35] considers all detour-free paths within its minimum bounding box, and gives an equal probability of being selected by the global router. Each net contributes the probability of occupying a track in the regions it may be routed through to the probabilistic usage of these regions. However, Westra's model [61] provides experimental proofs that in practice, most nets are routed with one or two bends. Therefore, the work [61] only considers the usages of L- and Z-shape routes, and then combines them based on probabilities weight extracted form real-life designs.

Fig. 2.5 show the difference between routing usage maps by these two models. Lou's model yields a high probabilistic usage at the center of the box spanned by a net (shown in Fig. 2.5 (a)); while Westra's model suggests a high probabilistic usage at the borders of the bounding box of the net (shown in Fig. 2.5 (b)).

Other probabilistic models are also proposed recently. For example, the work [46] uses diagonal-based model and assign probability to each grid in the


(a)

.01	.06	.14	.36	.50
.06	.12	.23	.29	.36
.14	.23	.26	.23	.14
.36	.29	.23	.12	.06
.50	.36	.14	.06	.01

(b)

.25	.29	.38	.46	.50
.29	.08	.08	.08	.46
.38	.08	.08	.08	.38
.46	.08	.08	.08	.29
.50	.46	.38	.29	.25

Figure 2.5: (a) Under Lou's model, all detour-free paths are considered. (b) Under Westra's model, only L- and Z-shapes are considered. [61]

same diagonal uniformly.

2.2.1.4 FastRoute

FastRoute [39][41] is a very fast global router which can be used as both an interconnect estimator and a traditional routing tool. FastRoute uses rip-up and reroute technique during global routing, and the main flow includes five phases:

1. Congestion map generation. The nets are decomposed into two-pin nets by Steiner minimal tree (FLUTE [10] construction). For each two-pin net, L-shape routing is applied to generate an initial congestion map.

2. Congestion-driven Steiner tree construction. If the routing path of a net passes though a congested edge, we have no way to simply eliminate the routing demand on the edge. Changing the Steiner tree topologies can give a lot of flexibility in avoiding routing congestion, which the widely used pattern routing and maze routing cannot achieve. By using the congestion-scaled distance instead of the original distance, the congestion-driven Steiner tree can reduce congestion significantly. Moreover, edge shifting is applied to do further improvement [39].

3. L- and Z-shape pattern routing. After decomposing nets based on their congestion-driven Steiner tree topology, L- and Z-shape pattern routing is applied to do rip-up and reroute.

4. Monotonic routing. For a two-pin net that spans $n \times m$ grids, L-shape pattern routing only considers 2 different paths, and Z-shape pattern routing only considers $m+n$ different paths. These limitations of pattern routing make it hard to find good routing paths. Monotonic routing is a trade-off between maze routing and pattern routing. The quality can be better than pattern routing, but the runtime is similar. Since all monotonic routing paths will not go out of the bounding box of the two-pin net, the total number of monotonic routing paths is $\binom{m+n-2}{m-1} = \frac{(m+n-2)!}{(m-1)!(n-1)!}$ [41].

5. Maze routing. Maze routing is the most popular technique used in global routing. Unlike original maze routing algorithm which is designed to find the shortest path connecting two pins in the presence of routing blockages, it is extend to multi-source multi-sink maze routing.

2.2.2 Congestion reduction techniques

Routability-driven placement problem has been studied in many previous works. The optimizations can be performed during (1) global placement, (2) detailed placement, and (3) the post-placement process. Table 2.2 summarizes these techniques. We will introduce several recent techniques in detail as follows.

2.2.2.1 Whitespace allocation

Several previous works try to distribute the routing demand evenly by reducing the cell densities in the congested regions. It is shown that by reducing

Table 2.2: Prior routability techniques

Category	Specific Techniques
In global placement	Movable nodes relocation [25][50][56][20] Cell inflation [17][18][23][29][4] Growing or shrinking placement regions [43] Macro block handling [12][20][21] Hierarchy design [11][20] Local placement refinement [42]
In detailed placement	Congestion aware cell swapping [29][42] Linear placement based on Steiner length in small windows [24][45]
Post-process	Network flow [57][59] Bloating and spreading cell by using pin density and congestion map [44] Module shifting by expanding GCell [66] Whitespace insertion or reallocation [32][64][45] Linear programming [33]

the densities of the cells on a chip, the routing demand will be distributed more sparsely [64] [32] [4] [44], and the routing congestion will be lowered.

The authors of [64] presents a whitespace allocation approach to allocate whitespace according to the congestion map. There are two steps: (1) allocate whitespace to each row, and (2) allocate whitespace to each bin within rows. Assume that there are n rows in the chip. Row congestion c_j is defined as the total congestion of the bins in row j, and row whitespace w_j is the total whitespace to be allocated to row j. Let W be the total whitespace of the chip. The relationship between w_j and W is shown in Equation (2.9).

$$\sum_{j=1}^{n} w_j = W \tag{2.9}$$

If there is a minimum row whitespace w_{min} constraint, the row whitespace can be assigned proportionally to the row congestion by Equation (2.10).

$$w_i = w_{min} + \frac{W - n \times w_{min}}{\sum_{j=1}^{n}(c_j - c_{min})}(c_i - c_{min}) \tag{2.10}$$

where c_{min} is the minimum row congestion among all the rows.

If both the minimum and maximum row whitespace are limited ($w_{min} \leq w_j \leq w_{max}, j = 1, \ldots, n$), a monotonic function $w_j = f(c_j)$ should be designed so that $w_i \leq w_j$, if $c_i \leq c_j$. The rows are first sorted according to their congestions in non-decreasing order ($c_1 \leq c_2 \leq \cdots \leq c_n$) . Hence, $w_{min} \leq w_1 \leq w_2 \leq \cdots \leq w_n \leq w_{max}$. Let $f(x)$ be in the form $f(x) = a_1 x^2 + a_2 x + a_3$, three constraints can be obtained (Equation (2.11)).

$$\begin{aligned} & a_1 c_1^2 + a_2 c_1 + a_3 = w_{min} \\ & a_1 c_n^2 + a_2 c_n + a_3 = w_{max} \\ & a_1 \textstyle\sum_{i=1}^{n} c_i^2 + a_2 \sum_{i=1}^{n} c_i + a_3 n = W \end{aligned} \tag{2.11}$$

The function $f(x)$ could be determined by solving Equation (2.11). Some adjustments are also discussed in the work [64] to make sure that the function is monotonic in the range $[c_1, c_n]$.

In the second step, it is reasonable to allocate whitespace to each $bin_{i,j}$ proportionally to the ratio of the congestion to the total congestion, i.e., $w_{i,j} = w_j c_{i,j} / c_j$. Other ratios can also be used, e.g., the ratio of the bin congestion square to the total square of the bin congestion.

Another method of whitespace allocation is proposed in [32]. A slicing tree based on the geometric locations of all the cells are first constructed, and this process is similar to a partitioning-based flow. The congestion level at each node of the tree are then estimated. According to the congestion, the cutline location at each node is adjusted in a top-bottom fashion to distribute the whitespace to two child nodes. Consider a region r with lower left conner (x_0, y_0), upper right coner (x_1, y_1), and the area of this region is $A_r = (x_1 - x_0)(y_1 - y_0)$. Assuming that the original cut is at $x_{cut} = (x_0 + x_1)/2$ in vertical direction. The total area of the cells on the left subregion r_0 and the right subregion r_1 are S_0 and S_1 respectively, and the corresponding congestion levels are c_0 and c_1. In order to keep the whitespace proportional to the congestion levels of two subregions, the new cut location x'_{cut} can be computed by Equation (2.12).

$$\begin{aligned} \gamma &= \frac{S_0+(A_r-S_0-S_1)\frac{c_0}{c_0+c_1}}{A_r} \\ x'_{cut} &= \gamma x_1 + (1-\gamma)x_0 \end{aligned} \tag{2.12}$$

At last, a detailed placer is applied to remove overlaps of cells and further reduce HPWL while preserving the whitespace distribution.

2.2.2.2 Cell inflation

CRISP [44] is an incremental placer. By using a fast global router instead of an inaccurate probabilistic congestion estimation, CRISP inflates and spreads cells in the congested regions iteratively. Fig. 2.6 shows an example of how CRISP inflates and spreads cells in a congested region. After spreading cells, the sizes of the cells will be reset to their original values. However, the technique of cell inflation applied at a later stage of the placement phase can only allow width expansion because the height must match with the row height of the standard cell. As a result, the horizontal routing congestion cannot be reduced. Moreover, by spreading cells horizontally, the demand of the horizontal tracks may even be increased [2].

Cell inflation performed during global placement is thought to be more flexible and powerful. In BonnPlace [4], cell areas in a congested region are expanded, and this technique is incorporated into a partitioning-based placer to affect whitespace allocation.

For cell i in a congested region, its inflated size will be $(1+b(i))\times s(i)$, where $s(i)$ is the original area of cell i, and $b(i)(\geq 0)$ is the inflation ratio. The initial value of $b(i)$ is proportional to the number of pins of cell i divided by $s(i)$. It is observed

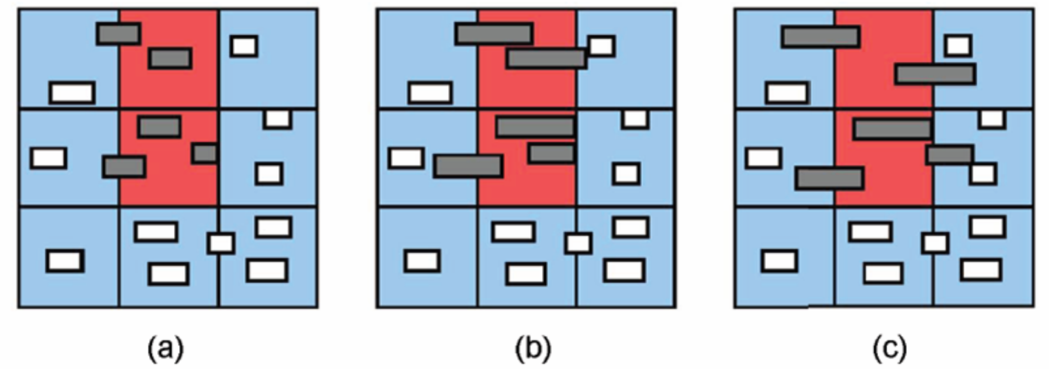


Figure 2.6: (a) Placement with congested region (b) CRISP inflates cells in these regions (c) Spread cells after inflation [44].

that small cells with many pins often cause routing problem when they are placed densely. During the placement process, $b(i)$ is updated according to the congestion estimation. In additional, $b(i)$ can be decreased if both the congestion estimation on the routing grid edges and the pin density are far away from the critical values. After cell inflation, BonnPlace [4] uses repartitioning method to move cells out of regions that are too full.

2.2.2.3 Congestion optimization in analytical function

Some previous works try to change the objective function in analytical placement to account for routability. By adding an additional force or cost to pull cells away from highly congested region to less congested region, we can reduce the routing congestion as well.

In NTUplace [21] (the version called Radiant in ISPD 2011 contest), when considering routing congestion, the placement problem is formulated by Equation (2.13).

$$\begin{aligned} \min \quad & W(x,y) \\ \text{s.t.} \quad & D_b(x,y) \leq M_b \text{ , for each bin } b \\ & E_e(x,y) \leq C_e \text{ , for each routing edge } e \end{aligned} \tag{2.13}$$

where $D_b(x,y)$ is the potential function that is the total area of movable blocks in bin b, M_b is the maximum area of movable nodes in bin b, $E_e(x,y)$ is the expected usage of routing edge e, and C_e is the total routing capacity of e. Note that $E_e(x,y)$ is a function of node positions. In the work [21], a 0-1 logic function is used to formulate the relationship between routing usage and node position. A Quadratic sigmoid function is then applied to smooth the $E_e(x,y)$ to $\widehat{E}_e(x,y)$, where $\widehat{E}_e(x,y)$ is the smoothed expected usage function of e. At last, the quadratic penalty method is used to transform the constrained optimization problem into an unconstrained problem as shown in Equation (2.14).

$$\min \lambda_1 \widehat{W}(x,y) + \lambda_2 \sum_b (\widehat{D}_b(x,y) - M_b)^2 + \lambda_3 \sum_e (\widehat{E}_e(x,y) - C_e)^2 \tag{2.14}$$

where λ_1, λ_2 and λ_3 are weigths for wirelength, cell-density and routability respectively.

Many other methods are proposed as well. Aplace [27] uses a probability-based approach to estimate congestion and incorporates this information in a logarithmic-sum-exponential (LSE) wirelength objective function. Kraftwerk2 [50] proposes a fast routing demand estimation method RUDY, and modifies the move force in their quadratic function to move cells to regions with lower routing demand. NTU-place [25] tries to remove overlaps between net bounding boxes to reduce routing congestion. Observing that net overlap removal may go too far since the bounding box of a net can only give a very rough estimation of its route, the approach in [11] uses fence force that utilizes the design-hierarchy information to group cells and uses local spreading force to reduce cell density in congested regions to balance between routability and wirelength during global placement.

2.2.2.4 Hierarchy design

Hierarchical design methodologies are usually employed in modern circuit design. It is expected that placing circuit elements that belong to the same design hierarchy level closely will have better circuit performance and routability [11][55]. It should be mentioned that the benchmarks used in the ICCAD 2012 contest of design hierarchy aware routability-driven placement [55] have the hierarchical information of the nodes.

NTUplacer [20] makes use of the hierarchy information in their multilevel framework for clustering in the coarsening stage. While the initial node locations are determined after the coarsening stage, clustering could help to preserve design hierarchies during the uncoarsening stage.

Before clustering, the work [20] uses macro-based hierarchy grouping. This grouping information will be considered in the subsequent clustering process. Since most macros are preplaced and fixed in modern mixed-size circuit designs [55], it is reasonable to consider both the hierarchy information and the positions of the macros simultaneously. According to the design hierarchy of the macros, macro hierarchical structure trees (MHSTs) are first constructed. Fig. 2.7 (a) shows an example of a MHST for a set of macros. The corresponding fixed layout of these macros is shown in Fig. 2.7 (b). Macros belonging to a MHST with similar shapes are grouped first (shown in Fig. 2.7 (c)), and other macros are grouped in a bottom-up manner according to the MHSTs (shown in Fig. 2.7 (d)). Although the macros on the upper-left and lower-right corners belong to the same hierarchy, they should not be identified as the same hierarchy group because of their large distance separation. Finally, for each macro, the cells that are at the same hierarchy with the macro will be grouped with it. For cells that are not at the same hierarchy with any macro will be grouped according to the design hierarchy information [11].

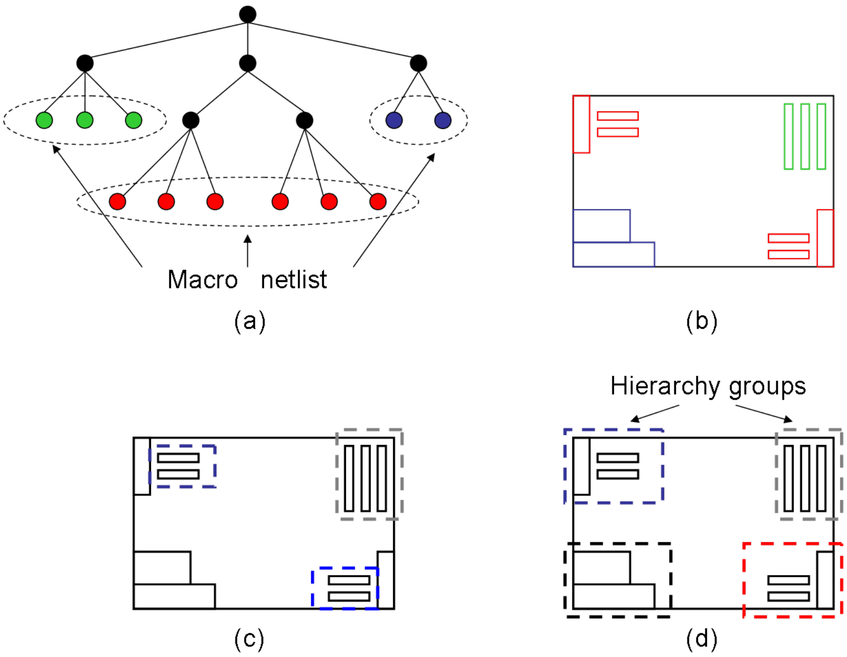


Figure 2.7: Macro-based hierarchy grouping [20].

During the clustering process, both the design hierarchy and the net connectivities should be considered. The affinity $A(v_i, v_j)$ between two nodes v_i and v_j is defined by Equation (2.15).

$$\begin{aligned} A(v_i,v_j) &= \frac{T_{i,j}}{|e_{i,j}|\cdot|a_i+a_j|} \\ T_{i,j} &= \begin{cases} \alpha\cdot k_{i,j} & \text{if } v_i,\ v_j \text{ are in the same hierarchy group} \\ k_{i,j} & \text{otherwise} \end{cases} \end{aligned} \tag{2.15}$$

where $|e_{i,j}|$ is the total number of pins of nets connecting v_i and v_j, a_i and a_j are the areas of v_i and v_j respectively, $T_{i,j}$ is the hierarchy factor, $k_{i,j}$ is the number of common hierarchy parts between these two nodes, and α is a factor set to 10 in [20].

2.2.2.5 Narrow channels handling

As illustrated in the work [20], many routing overflows occur along the macro boundaries, especially for narrow channels between macros. Placing too many cells in these narrow channels would cause routing overflow as shown in Fig. 2.8. Besides, cells placed in narrow regions have difficulty in escaping during routing.

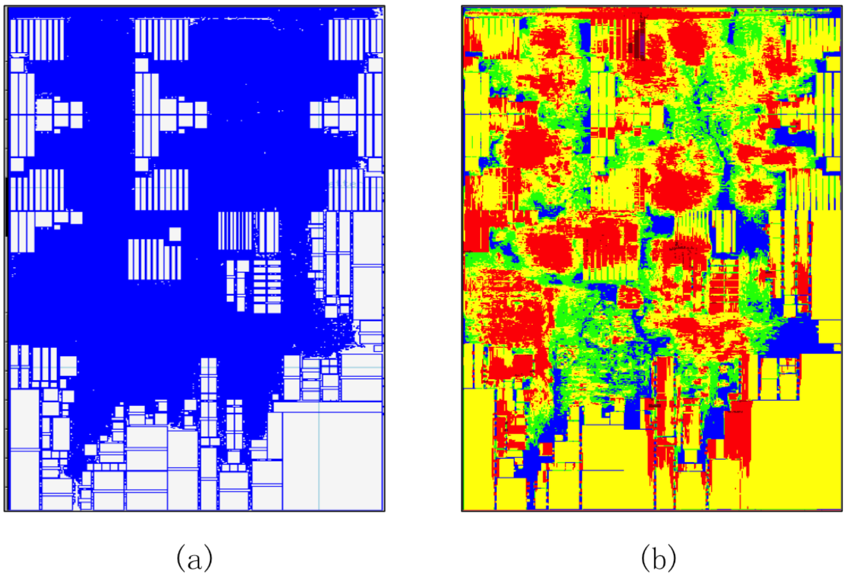


Figure 2.8: Illustration of narrow channels (superblue7 in DAC 2012 contest [52]). (a) The placement result given by Ripple. All movable cells are in blue color. (b) The corresponding global routing result from NCTUgr [34]. The narrow channels between macros are usually potential routing congested region (shown in red color) in (b).

In order to reduce the congestion caused by narrow channels, the work [12] chooses to inflate fixed macros based on their distance to neighboring fixed macros. For each fixed macro, a set of right-side macro neighbors whose distance is smaller than a right-threshold value are found. The sets of neighbors of its left-side, top-side and bottom-side can be found similarly. Four inflation ratios are then computed for the macro with respect to these four sides. For example, to completely block a narrow channel on the right of a macro, inflation ratio of the right-side of the macro should be equal to 100%.

NTUplacer4 [20] proposes a method that dynamically modifies the base potential around narrow channels in their analytical placer. They first detect the locations of narrow channels, and then change the potential function of the bins that reside on these narrow channels. Gaussian smoothing method [15] with a small σ (e.g. 0.5 in their implementation) are used to differentiate the potential function.

2.2.3 Case studies of routability-driven placers

2.2.3.1 SimPLR

Here, we first briefly introduce the framework of the traditional placer SimPL [30]. Then, the routability optimization techniques used in SimPLR [29][23] will be discussed.

SimPL [30] is a self-contained, flat, quadratic global placer. It is developed based on the lower-upper-bound framework as shown in Fig. 2.9. The two bound computations are invoked alternately. In the lower bound computation, the HPWL objective is represented as quadratic function (shown in Equation (2.3)) according to the Bound2Bound net model [49], and this quadratic function will be solved by the Conjugate Gradient (CG) decent method. In the upper-bound computation phase, a lookahead legalization algorithm based on geometric top-down partitioning and non-linear scaling is proposed to alleviate the overlaps between cells. The two bound computations are invoked alternately until the computations of these two bounds converge. Artificial two-pin pseudonets are introduced in the lower bound computation to consider the target positions of cells obtained in the upper bound computation.

During lookahead legalization (LAL), the chip is divided into uniform grids.

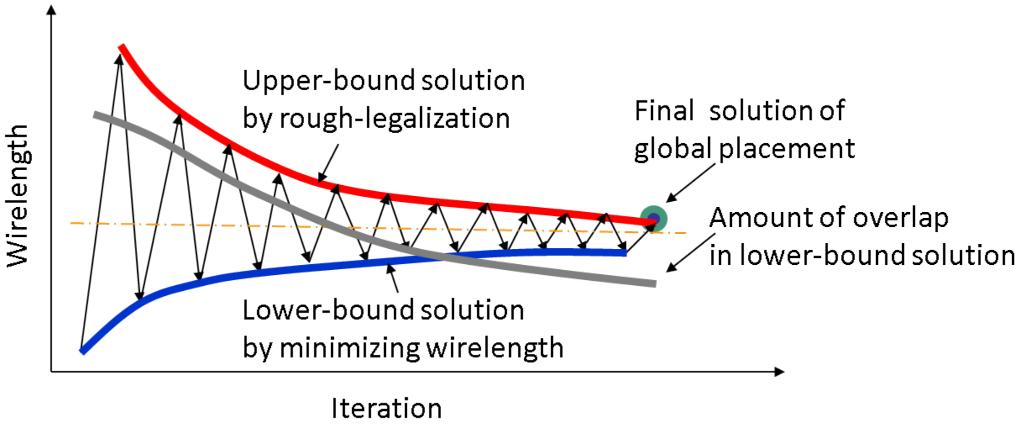


Figure 2.9: The upper-lower-bound framework.

Adjacent grids with significant amount of cell overlaps are clustered. LAL seeks to remove most of the overlaps (with respect to the grid) while preserving the relative ordering of the cell locations. For each cluster, the lookahead legalization is based on a top-down recursive geometric partitioning and non-linear scaling. Fig. 2.10 shows an example of lookahead legalization process.

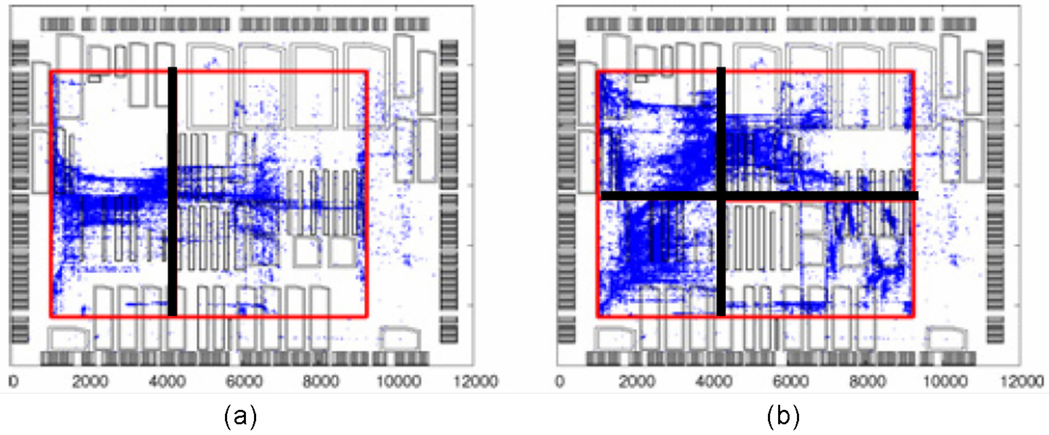


Figure 2.10: (a) Non-linear scaling is first performed in the x-direction (b) Subsequent non-linear scaling is applied in two sub-regions in the y-direction (Adaptec1). All movable cells are in blue color [30].

When considering routability, SimPLR applies several techniques, such as dynamic adjustment of target density, cell inflation, and so on. The SimPLR flow is shown in Fig. 2.11. BFG-R [22] is applied for lookahead routing.

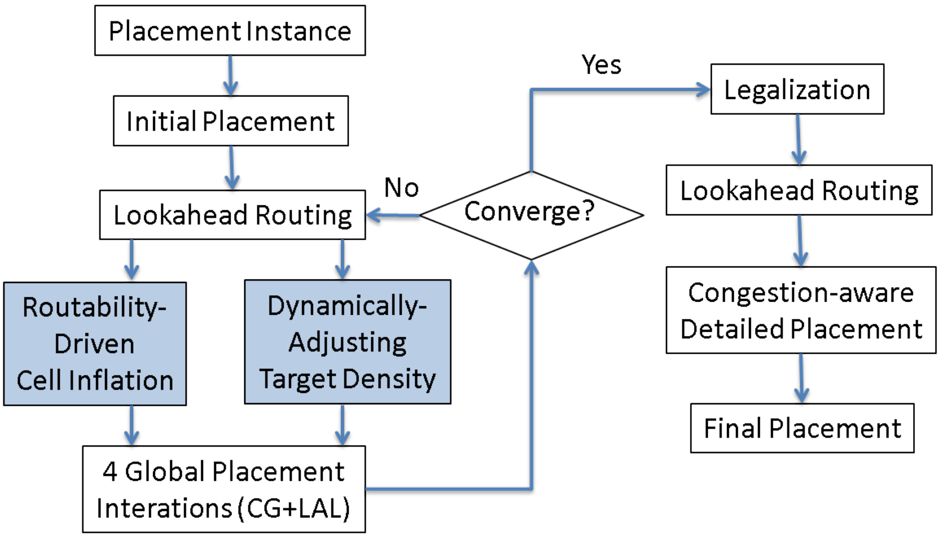

Placement Instance
Initial Placement
Lookahead Routing
Routability-Driven Cell Inflation
Dynamically-Adjusting Target Density
4 Global Placement Interations (CG+LAL)
Converge?
No
Yes
Legalization
Lookahead Routing
Congestion-aware Detailed Placement
Final Placement

Figure 2.11: The routability flow of SimPLR [29].

The work [29] dynamically adjusts the target density during global placement. The initial target density is set to:

$$\gamma_{init} = D_{ut} + \min\{\max\{\gamma_0 - D_{ut}, 0\%\}, \omega_D\} \tag{2.16}$$

where D_{ut} is the design utility (given), γ_0 is a prediction of a good target density, and ω_D is the target density lower bound. In the work [29], $\gamma_0 = 50\%$ and $\omega_D = 15\%$. After lookahead routing and cell inflation, the target density is updated by Equation 2.17.

$$\gamma = \min\{\frac{area(C_m)}{area(D) - area(C_f)} + \phi, 95\%\} \tag{2.17}$$

where C_m is the set of movable cells, C_f is the set of fixed cells, D is the design, the function *area* returns the total area of input including inflated cells, and ϕ is a number that is initially $\gamma_{init} - D_{ut}$, and increases by 1% when the routed wirelength increases according to the lookahead routing.

For routability-driven cell inflation, the work [23] argues that when inflated cells move outside the congested region, new coming cells have to be inflated again, and this process may waste whitespace without solving the root cause of congestion in the given region. This problem is also confirm by several industrial and academic studios.

The work [23] summarized three types of congestion:

- Cell-based congestion: caused by cell-to-cell interconnections.
- Local layout-based congestion: caused by static design properties, like blockages and routing capacities reduction.
- Remotely-induced layout-based congestion: caused by global nets which pass through the congested region.

In order to reduce cell-based congestion, we can resort to inflating the cells in the congested region. However, inflating too many cells or overinflate some cells may even worsen both the routability and wirelength. To ensure steady improvement, SimPLR [29] chooses to inflate cell in the top 5% most congested GCells by Equation (2.18)

$$\max\{\text{width}(cell)+1, 1+\theta(G)\cdot\Lambda(cell)\cdot\deg(cell)\} \tag{2.18}$$

where width($cell$) and deg($cell$) are the width and connectivity of $cell$ in a congested GCell, respectively. $\theta(G)$ is an adaptive function reflecting by the routability and difficulty of the design, and $\Lambda(cell)$ is the number of times $cell$ has been in a congested GCell.

In order to reduce layout-based congestion, the work [23] enforces non-uniform target densities in localized regions. The local layout-based congestion which is caused primarily by static constraints such as custom routing edge reduction, can be solved through locally injecting whitespace. The remotely-induced layout-based congestion, where congested GCells contain no standard cells but have a few routing tracks traversed by long nets, the non-uniform target density can be implemented by inserting fixed dummy cells at the center of every GCell and modifying their size based on congestion.

2.2.3.2 NTUplacer4

In the traditional placer NTUplacer [8], the placement region is first divided into uniform nonoverlapping bins. The global placement problem is formulated as a constrained minimization problem as shown in Equation (2.19).

$$\begin{aligned} &\min W(x,y) \\ &\text{s.t. } D_b(x,y) \leq M_b \text{ , for each bin } b \end{aligned} \tag{2.19}$$

where $W(x,y)$ is the wirelength function, $D_b(x,y)$ is the potential function that is the total area of movable blocks in bin b, and M_b is the maximum area of movable nodes in bin b. M_b is computed by $M_b = t_{density}(w_b h_b - P_b)$, where $t_{density}$ is target density value for each bin, w_b and h_b are the width and height of bin b, and P_b is the base potential equal to the pre-placed block area in bin b. M_b is a fixed value when all the positions of preplaced blocks are given and the bin size is determined.

The objective $W(x,y)$ is usually defined as the total HPWL of all nets, and the smooth function (Equation (2.5)) is used in NTUplacer. The potential function $D_b(x,y)$ is defined according to the overlaps of each bin and each block along the x and y directions, and the bell-shaped function is used to smoothen the density for each block [27].

Equation (2.19) can be solved by the quadratic penalty method shown in Equation (2.20).

$$\min \widehat{W}(x,y) + \lambda \sum_{b} (\widehat{D}_b(x,y) - M_b)^2 \tag{2.20}$$

where λ is a penalty factor, $\widehat{W}$ and $\widehat{D}$ are the smoothed wirelegnth and density functions. This unconstrained problem in Equation (2.20) can be solved by CG methods. NTUplacer uses multi-level framework (Section 2.1.2.2). During the uncoarsening stage, the placement problem is solved from the coarsest level to the finest level, where the solution for the current level provides the initial placement for the next level.

Routability optimization is considered in NTUplacer4 [20] (the version in ICCAD 2012 contest). The framework is shown in Fig 2.12. Three major techniques are proposed: (1) Design hierarchy identification (Section 2.2.2.4), (2) Narrow channel handling (Section 2.2.2.5), and (3) Net congestion optimization. In this section, we will introduce the congestion estimation and the net congestion optimization.

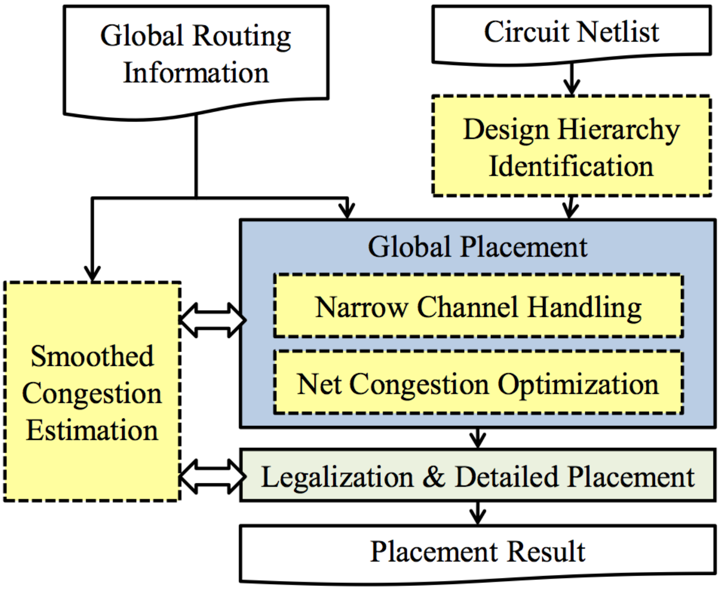

Global Routing Information
Circuit Netlist
Design Hierarchy Identification
Global Placement
Smoothed Congestion Estimation
Narrow Channel Handling
Net Congestion Optimization
Legalization & Detailed Placement
Placement Result

Figure 2.12: The overview of NTUplacer4 [20].

Since the exact routing is unknown during placement, routability is an abstract concept. L-shape probabilistic routing model is applied in NUTplacer4. The nets are first decomposed into 2-pin nets by a RSMT algorithm (FLUTE [10]), and then each 2-pin net is routed by upper-L and lower-L patterns with 50% probability for each direction. At last, Gaussian smoothing [15] is used to smooth the L-shape approximation model.

In the following, the technique of routing congestion optimization will be discussed. The form of constrained minimization problem is shown in Equation (2.21).

$$\begin{aligned} \min \quad & W(x,y) \\ \text{s.t.} \quad & D_b(x,y) \leq M_b \text{ , for each bin } b, \\ & C_b(x,y) \leq S_b \text{ , for each bin } b. \end{aligned} \tag{2.21}$$

where $C_b(x,y)$ is the net congestion of bin b, S_b is the total allowable routing area in bin b. Note that C_b is a function of block positions.

To evaluate the net congestion of bin b, the weighted perimeter-per-area ratio of a net e is defined by Equation (2.22)

$$d_e(x,y) = \frac{L_{x,e}(x,y) \cdot \sigma_v + L_{y,e}(x,y) \cdot \sigma_h}{L_{x,e}(x,y) + L_{y,e}(x,y)} \tag{2.22}$$

where $L_{x,e}$ and $L_{y,e}$ are the respective horizontal and vertical bonding box lengths of net e, and σ_h and σ_v are respectively the average vertical and horizontal wire pitches. When minimizing $d_e(x,y)$, square bounding box is preferred.

The overlaps between a net e and a bin b is computed by Equation (2.23)

$$\phi_{b,e}(x,y) = o_h(b,e) \cdot o_v(b,e) \tag{2.23}$$

where $o_h(b,e)$ and $o_v(b,e)$ are the horizontal and vertical overlaps between net e

and bin b.

The net congestion of a bin b is defined by Equation (2.24)

$$C_b(x,y) = \sum_e (d_e(x,y) \cdot \phi_{b,e}(x,y)) \tag{2.24}$$

The quadratic penalty method is used to solve Equation (2.21) by transforming the constrained optimization problem into an unconstrained problem shown in Equation (2.25).

$$\min \lambda_1 \widehat{W}(x,y) + \lambda_2 \sum_b (\widehat{D}_b(x,y) - M_b)^2 + \lambda_3 \sum_e (\widehat{C}_b(x,y) - S_b)^2 \tag{2.25}$$

where λ_1, λ_2, and λ_3 are weights for wirelength, density, and net congestion, respectively, and $\widehat{C}_b$ is the smoothed net congestion of a bin b.

2.2.3.3 mPL12

The framework is based on analytical placement approaches [21][6][38]. Since we have already briefly introduced these techniques in Section 2.1.2 and Section 2.2.3.2, we focus on the issues of routability optimization [12] in the following.

To estimate the routing congestion, the chip is divided into uniform tiles. Multipin nets are first decomposed into two-pin nets by FLUTE [10]. The routing supply and demand of each tile in the horizontal and vertical directions are computed separately. Similar to Ripple 1.0 [18], the routing congestion value $CongestionH(i)$ of tile i in the horizontal direction are computed by Equation (2.26)), and the congestion in the vertical direction can be obtained similarly.

$$\begin{aligned}
&SupplyH(i) = (TileWidth)(TileHeight) - BlockageH(i),\\
&DemandH(i) = \textstyle\sum_e \frac{Ovlp(i,e)WireH(e))}{WidthBB(e)HeightBB(e)},\\
&CongestionH(i) = \frac{SupplyH(i)-DemandH(i)}{SupplyH(i)},
\end{aligned} \tag{2.26}$$

where $TileWidth$ and $TileHeight$ are the width and height of each tile, $BlockageH(i)$ is blocked area of horizontal tracks occupied by routing blockages in tile i, $WidthBB(e)$ and $HeightBB(e)$ is the width and height of the bounding box of two-pin net e, $Ovlp(i,e)$ is the overlapping area between tile i and the bounding box of net e, and $WireH(e)$ is the horizontal wire area of net e.

Cell inflation is used to alleviate congestion with the similar inflation pattern in Ripple 1.0 [18]. To further reduce congestion, three techniques are proposed: (1) blocking narrow channels by inflating fixed macros that create such channels (discussed in Section 2.2.2.5), (2) inserting dummy cells inside regions with low density of fixed macro, and (3) pre-placement inflation. Here, we will introduce the techniques (2) and (3).

In some designs, the majority of macros are located at the periphery, and the existence of the regions with large amount of whitespace may become congested during placement. To avoid this type of congestion, dummy cells are inserted inside the large empty regions on the chip. To identify such regions, a coarse grid is applied to determine bins with reduced fixed macro density. And then, dummy cells are inserted in these particular bins.

Two kinds of pre-placement inflation are proposed in [12]:

1. GTL(Group of Tangled Logic)-based inflation. The GTL metric [26] is applied to detect tangled logic structures in a netlist. Let $T(C)$ be the net cut of cell cluster C, $|C|$ be the umber of cells in C, A_C be the average pin count

of cells in C, A_G be the average pin count of all cells, and p be the Rent exponent. The GTL score of C is defined by Equation (2.27). The GTL score curve of C is illustrated in Fig. 2.13 and corresponds to the growth of C. During the growth of a cluster from a seed cell, the score of the cluster is modified by iteratively ding highly connected neighbors. The curve contains a distinct trough if a tangled logic structure appears during the growth. The position of the trough indicates when the cluster growth has reached the logic structure with the most tangled logic. The allocation of whitespace among the detected tangled logic structures is proportional to their respective weights as shown in Equation (2.28), where $TroughWidth(C)$ correspond to the trough width of the GTL score curve of C. If the GTL score curve of C has a larger trough width, it has a more distinctive trough and C has more tangled logic structure. The wight of C is based on the observation that clusters with a large number of cells and small area are more likely to be congested and should be inflated.

$$GTL(C) = \frac{T(C)}{A_G |C|^{p \frac{A_C}{A_G}}} \tag{2.27}$$

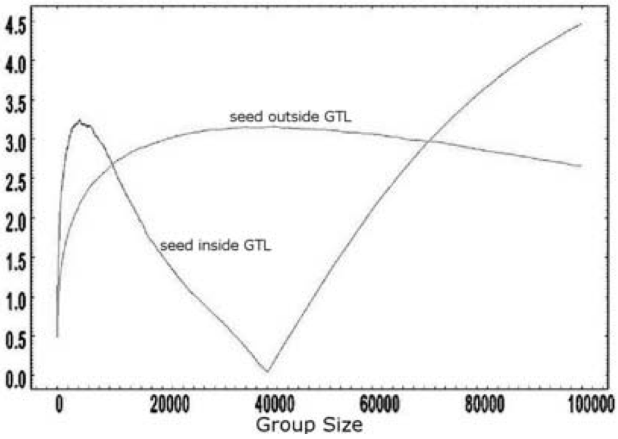


Figure 2.13: An example of the GTL-score curve [26].

$$Weight(C) = TroughWidth(C)\frac{|C|^2}{Area(C)} \tag{2.28}$$

2. Pin density-based inflation. This process is trying to minimize the maximum pin density. The cells are first sorted in the order of decreasing pin density, then the process determines how many of these cells can be inflated within the given whitespace budget, so that all inflated cells have the same pin density d_{max}. Let p_i be the number of pins of cell i, A_i and A_i' be the original and inflated area of cell i, and W be the whitespace allocated for cell inflation. The inflated area $\{A_i'\}$ is the variable in the following optimization problem shown in Equation (2.29).

$$\begin{aligned} \text{min} \quad & \max_i \{p_i/A_i'\} \\ \text{s.t.} \quad & A_i \le A_i',\ \forall i, \\ & \textstyle\sum_i (A_i' - A_i) = W. \end{aligned} \tag{2.29}$$

This problem can be solved optimally by four steps:

Step 1: Sort cells in descending order of their pin density $s.t. p_i/A_i \ge p_{i+1}/A_{i+1}, \forall i$.

Step 2: Find the largest index k $s.t.$ $I_k \le W \le I_{k+1}$, where $I_k = A_k(\sum_{i=1}^{k} p_i)/p_k - (\sum_{i=1}^{k} A_i)$.

Step 3: Compute the maximum pin density after inflation $d_{max} = (\sum_{i=1}^{k} p_i)/(W + \sum_{i=1}^{k} A_i)$.

Step 4: Compute the inflated area $A_i' = p_i/d_{max}(i \le k)$.

The cells are first sorted in descending order of their pin density (Step 1). Then, the target pin density are found by iteratively budgetting whitespace to cells until there is no whitespace left (Step 2). The cells whose pin density are larger than the target density will be inflated (Step 3-4). The proof of this algorithm is given in the work [12].

CHAPTER 3

Routability-driven global placement

Contents

The aim of this chapter is to propose several effective algorithms in the global placement of Ripple. In order to address routability problem, we first do lookahead routing analysis to identify congested regions. We then discuss the method to gradually move cell for routability. A robust and effective way to obtain the inflation ratio for cells in the congested regions, especially avoiding over-inflation when the congestion is very high. Besides, a method to spread the cells after inflation will also be discussed.

The rest of this chapter is organized as follows. In Section 3.2, we review related preliminaries. Section 3.3 gives an overview of our placer Ripple. Section 3.4 explains the congestion estimation method to computer congestion maps quickly. Section 3.5 analyzes the cell inflation and spreading during placement.

3.1 Problem formulation

Given a netlist $N = (E,V)$ with nets E and nodes V, the routing resource supplied in upper-metal layer (refer to Section 1.2), a routability-driven placer is required to compute the positions of the movable nodes (cells) to minimize the routing congestion and wirelength.

The main purpose of global placement is to distribute the cells evenly over the placement region. The major goal is to trade off between the congestion measure and the HPWL. As we want to maintain a global view, the global placement pays more attention to the relative positions among cells globally, and the layout may have some degree of overlapping between nodes.

HPWL is an objective used by many traditional placers. The HPWL function is usually approximated by a differentiable function, such as the quadratic objective shown in equation (3.1).

$$f(\vec{x},\vec{y}) = \sum_{i,j \in V} w_{i,j}((x_i - x_j)^2 + (y_i - y_j)^2) \tag{3.1}$$

where $\vec{x}$ and $\vec{y}$ are coordinate vectors of the cell locations, and $w_{i,j}$ is the connectivity weight between cells i and j. In our implementation, B2B net model [49] (introduced in Section 2.1.2) is applied to calculate the connectivity weight.

Since nets have not been routed yet during placement, and different routers have different routing results, the routing congestion is usually estimated by some models or simplified global router (discussed in Section 3.3). During global placement, we try to reduce the congestion measure obtained by congestion estimation.

3.2 Overview

Ripple is a flat placer with a lower-upper-bound framework [30] as shown in Fig. 3.1. In the lower bound computation, the quadratic objective shown in equation (3.1) is applied to minimizing HPWL. In the upper bound computation, we spread cells to roughly remove overlaps between cells and alleviate routing congestion. The two bound computations are invoked alternately until the computations of these two bounds converge.

The process of upper bound computation is shown in Fig. 3.2. We apply Simplified FastRoute with pin density consideration to analyze congestion of intermediate layouts. This global router not only give a more accurate congestion map comparing with previous probabilistic estimation, but also provide routing solutions for nets, which enables us to use the strategy of routing path-based cell inflation and spreading to alleviate congestion. It is known that the congestion of a region can be caused by three types of nets: (1) local nets connecting cells within the region, (2) semi-global nets coming into/out of the region, and (3) global nets passing through the region without connecting any cell inside the region. Different from many previous works that only spread cells in congested regions, the cells whose connections passing through congested regions will be inflated as well. Therefore, the congestion due to various types of nets can be alleviated effectively. A global placement solution is generated when the congestion and HPWL obtained by the lower and upper bound computations converge.

After a number of iterations (>30 iterations in our implementation), the congestion and HPWL will nearly converge. However, several congested clusters caused by tangled logic may appear. The cells of tangled logic may not be spread sparsely enough, even if we have already used the technique of cell inflation and spreading. Therefore, those cells in the tangled logic should be identified and made to spread more sparsely. Section 3.4.5 gives further discussion about this issue.

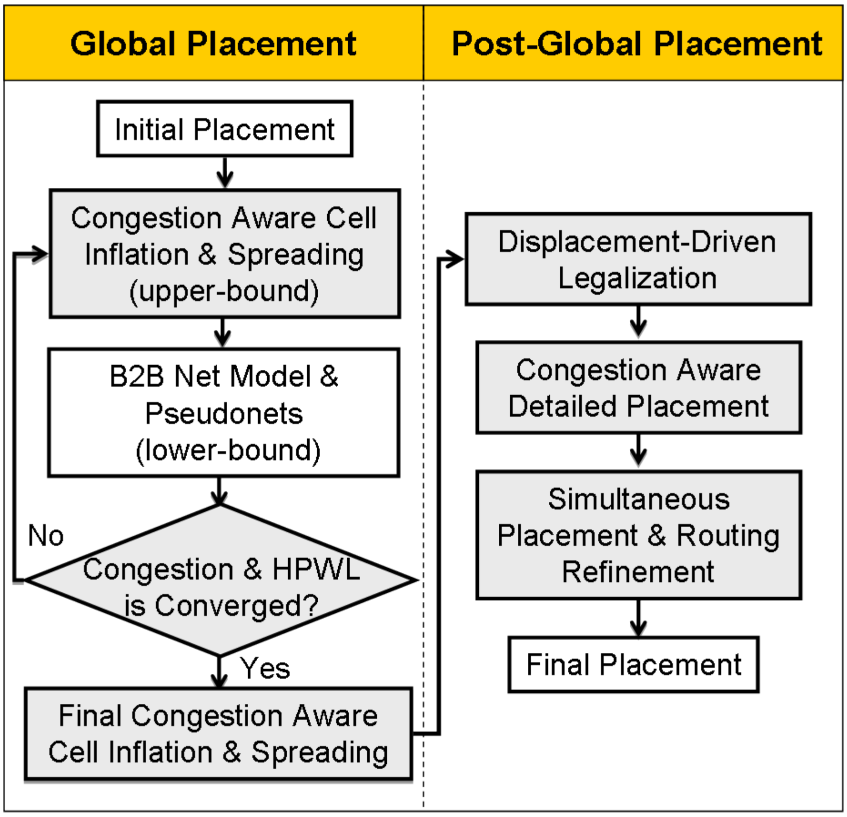

Global Placement
Post-Global Placement
Initial Placement
Congestion Aware Cell Inflation & Spreading (upper-bound)
B2B Net Model & Pseudonets (lower-bound)
No
Congestion & HPWL is Converged?
Yes
Final Congestion Aware Cell Inflation & Spreading
Displacement-Driven Legalization
Congestion Aware Detailed Placement
Simultaneous Placement & Routing Refinement
Final Placement

Figure 3.1: The framework of our routability-driven placer.

3.3 Congestion estimation

Congestion estimation is very important when considering routability. During placement, many intermediate layouts will be generated and evaluated by a con-

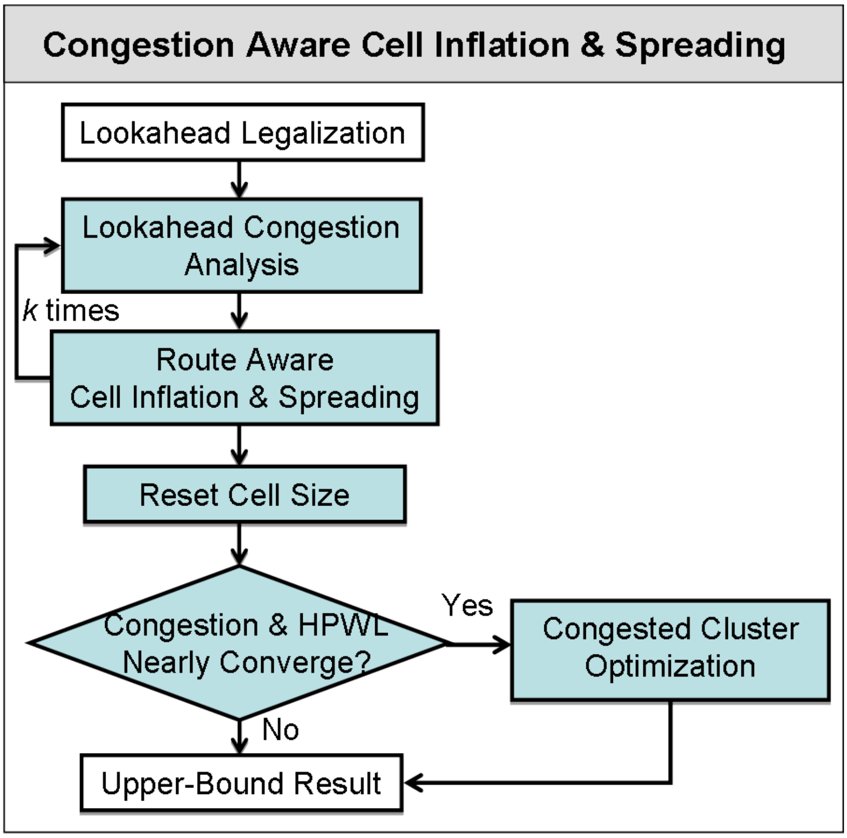
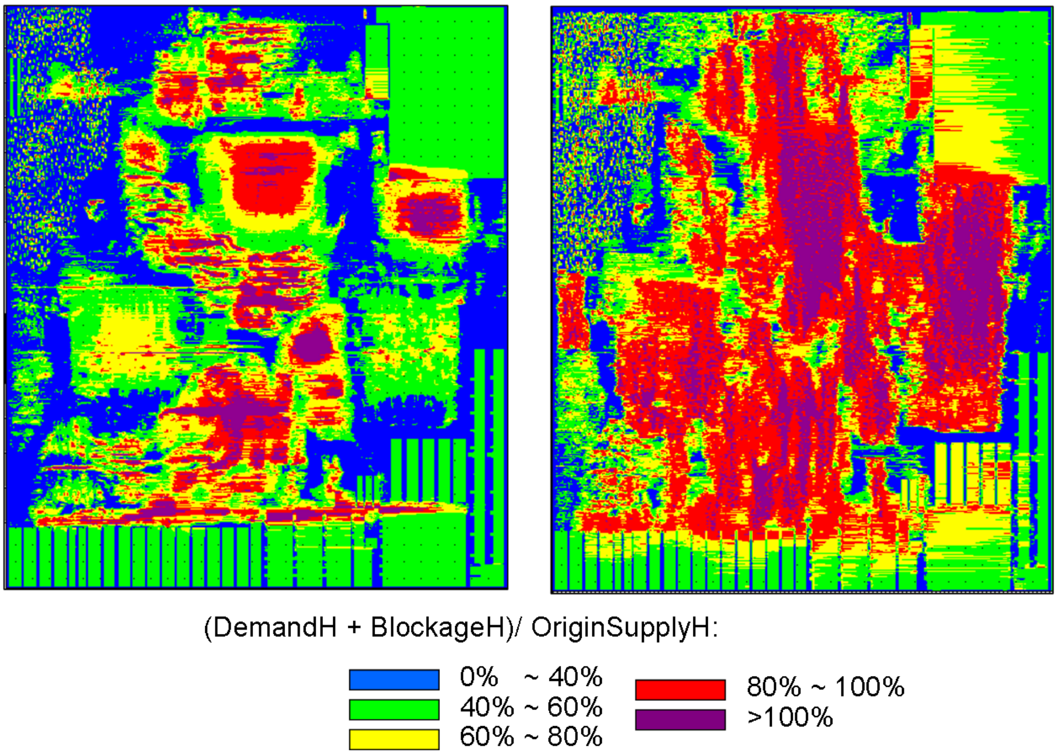

Congestion Aware Cell Inflation & Spreading
Lookahead Legalization
Lookahead Congestion Analysis
k times
Route Aware Cell Inflation & Spreading
Reset Cell Size
Congestion & HPWL Nearly Converge?
Yes
Congested Cluster Optimization
No
Upper-Bound Result

Figure 3.2: The flow of congestion-aware cell inflation and spreading.

gestion estimator. According to these estimations, the placer will make proper adjustment to improve congestion. Different kinds of probabilistic congestion estimations [28][50][61] [46] are proposed to capture the behavior of routing and perform congestion analysis. Recently, some global routers, like BFG-R [22] and FastRoute [39], are even used in placement for congestion estimation purpose. Using a router to do estimation is more accurate, and the routing paths are made available. However, the runtime of a router is usually much longer than that of those probabilistic methods. If a placer needs to make use of the routing path information or the number of calls to the estimator is not large, using a router directly may be more proper. On the other hand, if a placer needs to call the estimator frequently, a faster probabilistic method will be a better choice. Therefore, the selection of the



(a) HPWL estimation

(b) Congestion map from the global router coalesCgrip

Figure 3.3: The congestion estimation of HPWL is too optimistic. Both congestion maps are for the horizontal direction (superblue12).

congestion estimator depends on when and how the congestion information will be used during the placement process.

We tried various congestion estimation methods in Ripple. Section 3.3.1 and Section 3.3.2 explain how to do probabilistic estimation and apply simplified global router for congestion analyze respectively.

3.3.1 Probabilistic estimation

Our congestion estimation is obtained by extending and generalizing RUDY [50]. RUDY uses the HPWL estimation directly which is indeed much shorter than the actual wirelength. Therefore, it will often under-estimate the real routing conges-

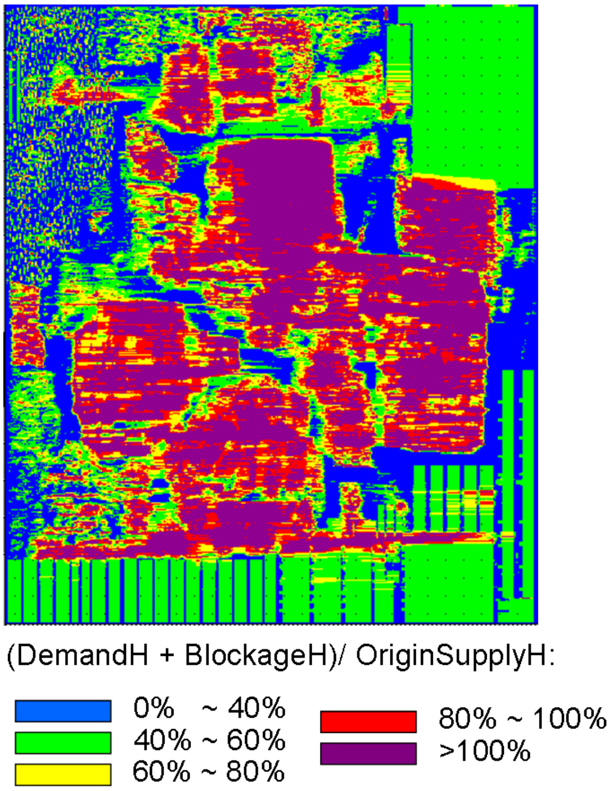


Figure 3.4: The congestion estimation map in the horizontal direction (superblue12) obtained with our proposed enhancements.

tion a lot as shown in Fig. 3.3(a) by comparing with Fig. 3.3(b) that is obtained from the global router coalesCgrip [48]. To cope with this, instead of using HPWL directly, multi-pin nets are first decomposed into two-pin nets by using the rectilinear minimum spanning tree (RMST) method. Shared tiles between subnets of the same net will be handled in such a way to avoid double counting. The routing demand is computed in the horizontal and vertical directions separately. We will discuss below the computation of the horizontal demand *DemandH*, while the vertical demand *DemandV* can be obtained similarly.

The layout is divided into uniform routing tiles whose width and height are TileSizeX and TileSizeY respectively. Consider a two-pin net p connecting from $pin1(x_1,y_1)$ to $pin2(x_2,y_2)$ shown in Fig. 3.5. The width of the bounding box of net

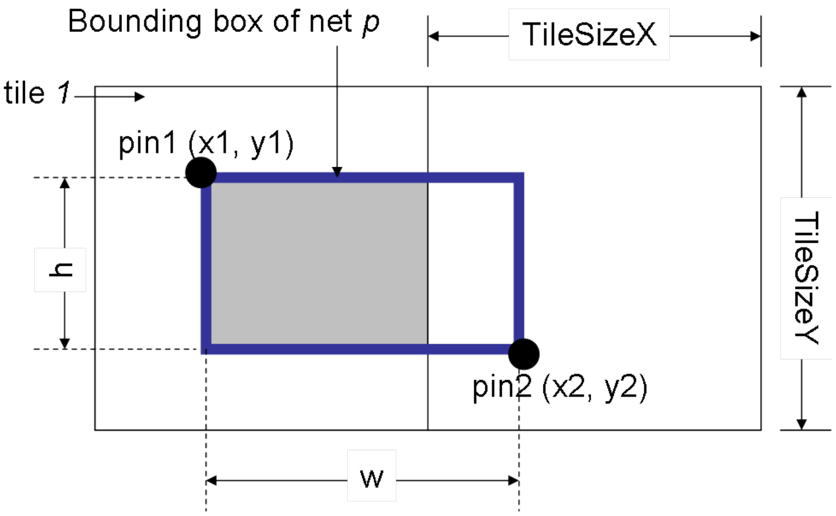


Figure 3.5: The routing demand of a two-pin net in tile is computed according to the overlapping ratio between the overlapped region (grey) and the bounding box of the net.

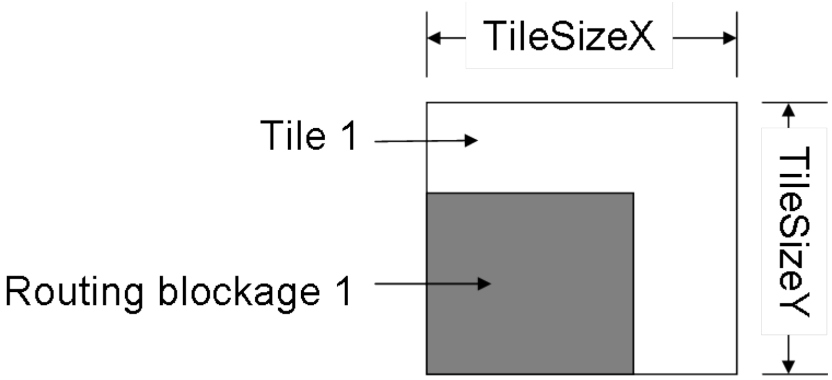


Figure 3.6: The routing supply of tile 1 has to exclude the blocked resources (region in grey).

p is $w = \max\{x_1, x_2\} - \min\{x_1, x_2\}$, and the height is $h = \max\{y_1, y_2\} - \min\{y_1, y_2\}$. The horizontal wirelength is the width of the bounding box. The *DemandH* due to p for each tile overlapping with the bounding box of p is computed as follows:

$$\begin{aligned}
DemandH &= \frac{ConnInterArea}{BoundingBoxArea} \times WireAreaH \\
WireAreaH &= (\max\{x_1, x_2\} - \min\{x_1, x_2\}) \times WireSpaceH \\
WireSpaceH &= \frac{TileSizeY}{TotalTracksH} \\
TotalTracksH &= \sum_{i=1}^{LayerNum} HTrack(i)
\end{aligned} \tag{3.2}$$

where *BoundingBoxArea* is the bounding box area of this connection, *ConnInterArea* is the intersection area between the tile and the bounding box, *WireSpaceH* is the area of one unit of wire in the horizontal direction, *WireAreaH* is the horizontal wire area of this net which can be obtained by multiplying the horizontal wirelength by *WireSpaceH* and *TotalTracksH* is the total number of horizontal tracks (counting all metal layers) in a tile. In Fig. 3.5, the *BoundingBoxArea* is $w * h$ and the *ConnInterArea* with tile 1 is the area of the region in gray. The horizontal wire area *WireAreaH* is equal to the wirelength w multiplied by the average horizontal wire width *WireSpaceH*. The *DemandH* of tile 1 is equal to the overlapping ratio *ConnInterArea*/*BoundingBoxArea* multiplied by the wire area *WireAreaH*.

The supply of a tile in the horizontal direction *SupplyH* can be computed as follows:

$$\begin{aligned}
SupplyH &= OriginSupplyH - BlockageH \\
OriginSupplyH &= TileSizeX \times TileSizeY \\
BlockageH &= \sum_{i \in Btile} BInterArea(i) \times BRatioH(i) \\
BRatioH(i) &= \frac{\sum_{j \in BlockLayer(i)} HTrack(j) \times (1 - BlockPorosity)}{TotalTracksH}
\end{aligned} \tag{3.3}$$

where $Btile$ is the set of routing blockages overlapped with the tile, $BInterArea(i)$ is the intersection area between routing blockage i and the tile, $BRatioH(i)$ is the ratio of horizontal tracks being occupied by the routing blockage i and $BlockageH$ is the routing resource occupied by all the routing blockages overlapped with the tile in the horizontal direction. If $SupplyH < DemandH$, this tile is congested. As shown in Fig. 3.6, tile 1 is overlapped with blockage 1 (grey color). $Btile$ of tile 1 is $\{blockage1\}$ and the blockage area in tile 1 is $BInterArea(1)$ (area of the grey region). The value of $BRatioH(1)$ is obtained by computing the ratio between the blocked routing tracks of blockage 1 and the total routing tracks of all layers. $OriginSupplyH$ of tile 1 is equal to $TileSizeX \times TileSizeY$. Note that the routing supply $SupplyH$ of tile 1 has to exclude the blocked resource due to blockage 1 which is equal to $BInterArea(1) \times BRatioH(1)$.

As mentioned above, we will handle the shared tiles of those subnets belonging to the same net carefully in order not to double count the routing demand of the same net mistakenly. As illustrated by the example in Fig. 3.7, using RMST, this four-pin net will be decomposed into three 2-pin subnets which are $a-b$, $a-c$ and $c-d$. Since both subnets $a-b$ and $a-c$ will increase the routing demand of tile 1, the demand of this net on tile 1 will be counted twice. In order to estimate more accurately, the $DemandH$ of tile 1 due to this net will be equal to the maximum $DemandH$ due to subnet $a-b$ and subnet $a-c$. Similarly, since tile 3 and tile 4 are overlapped by more than one subnet, the $DemandH$ of tile 3 and tile 4 will be equal to the maximum routing demand of their overlapping subnets respectively.

Fig. 3.4 shows the congestion map in the horizontal direction for superblue12 obtained by our method. We can see that more accurate estimation can be obtained by comparing Fig. 3.4 with the simple HPWL approach shown in Fig. 3.3(a).

Our estimation is probabilistic and the nets are not detoured if they go through

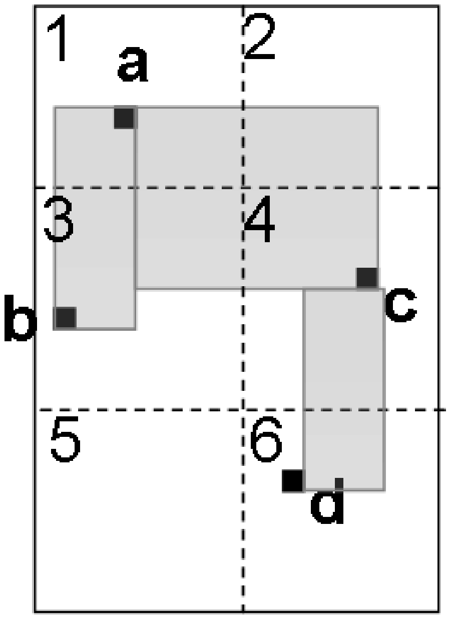


Figure 3.7: An example of repeated counting the routing demand. After decomposing a four-pin net into three two-pin nets: a-b, a-c and c-d, tile 1, tile 3 and tile 4 will overlap with more than one subnet. The routing demands of tile 1, tile 3 and tile 4 due to this net will be double without proper adjustment.

congested regions. Therefore, this estimation method may over-estimate overflow. However, without detouring nets, we can directly identify the congested regions and resolve the problem by relocating cells. We can also dynamically deal with the over-estimation problem which will be discussed in section 3.4. [1]

3.3.2 Lookahead routing analyze

In order to optimize routability while preserving wirelength, a lookahead routing is invoked during global placement. Unlike previous congestion estimations that only report congestion map, our lookahead routing analysis reports both routing congestion and interconnection paths. Besides, our lookahead routing accounts for

[1] Actually, we tried other probabilistic methods [46][61] and the Steiner tree approach (using FLUTE [9]) to do estimation. In our implementation, the congestion maps and the final results are similar. Since the method we proposed in this paper is faster and simpler, we use it in Ripple.

pin density as well.

The chip is divided into uniform GCells whose width and height are $TileSizeX$ and $TileSizeY$ respectively. Each pair of adjacent GCells have an edge connecting them. In our lookahead routing analysis, the routing supply and demand are computed in the horizontal and vertical directions separately. We will discuss below the computation of the horizontal supply and demand, while the vertical supply and demand can be obtained similarly.

3.3.2.1 Routing supply calculation

The routing supply of a GCell edge is equal to the total number of tracks supplied in all metal layers. Since different metal layers have different wire widths and wire spacings, we need to sum up the number of tracks of each metal layer. The default supply of a horizontal GCell edge e is obtained by equation (3.4).

$$DefaultSupplyH_e = \sum_{l=1}^{NumLayer} \frac{TileSizeY}{MinWidth_l + MinSpace_l}, \tag{3.4}$$

where $MinWidth_l$ and $MinSpace_l$ are the minimum wire width and spacing on metal layer l, and $NumLayer$ is the number of metal layers.

Since there are many fixed routing blockages occupying the routing resources on the metal layers, the supply of GCell edges need to exclude those blocked routing resource. In order to calculate the number of blocked tracks, we first obtain the union of all the blockages on a GCell edge. Then, we obtain the number of blocked tracks of each blocked region separately. As shown in equation (3.5), the supply of a horizontal GCell edge e is equal to $DefaultSupplyH_e$ minus the number of blocked tracks of e.

$$\begin{aligned} SupplyH_e &= DefaultSupplyH_e - BlockTrackH_e \\ BlockTrackH_e &= \sum_{l=1}^{NumLayer} \frac{BlockLength_{e,l}}{MinWidth_l + MinSpace_l} \times (1-p), \end{aligned} \tag{3.5}$$

where $BlockTrackH_e$ is the number of blocked tracks of edge e, $BlockLength_{e,l}$ is the length of the blocked region in the GCells connected by e on metal layer l and p is the porosity of the routing blockage (zero implies complete blockage).

The calculation of the routing supply is demonstrated in Fig. 3.8. In this example, $TileSizeY = 40$. Let both the minimum wire width and spacing be 2, and porosity p is 0. The $DefaultSupplyH_{e_1}$ and $DefaultSupplyH_{e_2}$ are both 40/(2+2)=10. Since two blockages overlap with GCells connected by e_1 and the total blocked length is 32, $BlockTrackH_{e_1}$ is 32/(2+2)=8 and $SupplyH_{e_1}$ is 10-8=2. Similarly, $BlockTrackH_{e_2}$ is 16/(2+2)=4 and $SupplyH_{e_2}$ is 6.

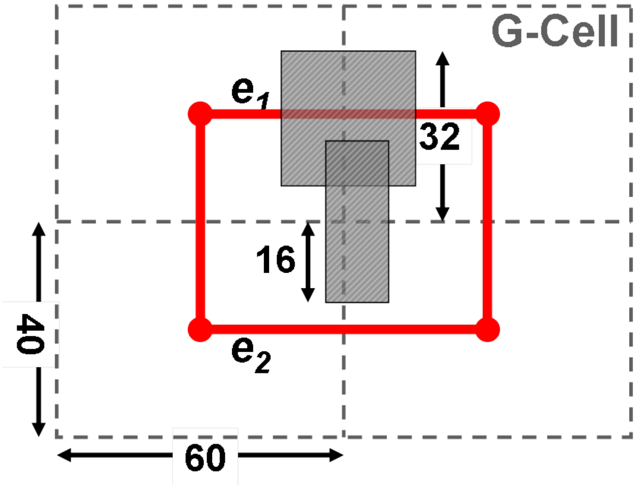


Figure 3.8: The GCells connected by horizontal edge e_1 and e_2 are overlapped with routing blockages. The supply of e_1 and e_2 have to exclude the blocked routing resources.

3.3.2.2 Routing demand calculation

We use FastRoute [39] to do routing congestion analysis. In order to check the overflow fast, we simplify FastRoute by removing the steps of monotonic routing and maze routing. Since global routing captures the wires that pass between GCell edges, internal congestion of GCells is ignored [44]. Therefore, we need to consider pin density within GCells to account for local congestion when computing the routing demand before invoking Simplified FastRoute.

In this Simplified FastRoute, multi-pin nets are decomposed into two-pin subnets by a congestion-driven method [39]. Then, a rip-up and reroute process will be invoked by using L/Z shape pattern routing. According to the current congestion map, if a net passes through congested regions, it will be rerouted. Since local peaks of pin density within GCell often cause routing congestion [44], pin density is added when constructing the initial congestion map. The demand of pin density on GCell edge e is obtained by equation (3.6).

$$PinDemandH_e = (PinNumL + PinNumR) \times \beta, \tag{3.6}$$

where *PinNumL* and *PinNumR* are the number of pins in the left and right GCells connected by e and β is the pin density factor. In our implementation, $\beta = 0.025$ at the beginning iterations and $\beta = 0.05$ when the congestion and HPWL nearly converge and further optimization step (section 3.4.5) is to be invoked.

After finishing Simplified FastRoute, we obtain the routing demand of each GCell edge e. The horizontal demand of e is shown in equation (3.7).

$$DemandH_e = PinDemandH_e + TrackH_e, \tag{3.7}$$

where $TrackH_e$ is the number of tracks passing through e which is obtained from Simplified FastRoute.

3.4 Congestion-based cell movement

In cell inflation, we need to answer three important questions. Where is cell inflation used? What is the inflation ratio? How to spread the cells? Unlike previous works [4][19][44], we consider these three issues as a whole for the horizontal and vertical directions separately. By applying our cell inflation and spreading strategy in the horizontal and vertical directions alternately for several times (six times in our implementation) in each iteration of global placement, the routing congestion in both directions will be alleviated steadily.

3.4.1 Cell inflation ratio calculation

After obtaining the routing demand as described in the section 3.3, the cells in the congested regions will be inflated. The following discussion is based on the horizontal direction while the vertical direction can be handled similarly.

In our implementation, we determine the associated tile of a cell based on the position of the cell center. In spite of the inaccuracy of ignoring other overlapping tiles, the discrepancy brought on average is not large since the sizes of the cells are usually much smaller than that of the tiles. For a congested tile $tile_{i,j}$, the cells in it will have the same inflation ratio. The values of *DemandH* and *SupplyH* are obtained as described in section 3.3. The tiles that with a *DemandH* larger than its *SupplyH* will be added into a list called $InflationList$. The inflation ratio $InflateRatio_{i,j}$ for the cells in $tile_{i,j}$ will be computed by equation 3.8. We will also limit the maximum inflation ratio (e.g., 2.5) to avoid over-inflation.

$$InflateRatio_{i,j} = DemandH/SupplyH \tag{3.8}$$

The new size $NewArea_k$ of a cell k in $tile_{i,j}$ after inflation is computed by equation 3.9.

$$NewArea_k = CellArea_k \times InflateRatio_{i,j} \tag{3.9}$$

where $CellArea_k$ is the original cell area of cell k.

According to the new sizes of the cells in each tile of the list $InflationList$, we will budget a total inflation area $TotalInflation$ according to equation 3.10.

$$\begin{aligned} TotalInflation &= \textstyle\sum_{tile_{i,j} \in InflationList} Inflation_{i,j} \\ Inflation_{i,j} &= \textstyle\sum_{cell_k \in tile_{i,j}} (NewArea_k - CellArea_k) \end{aligned} \tag{3.10}$$

where $Inflation_{i,j}$ is the total inflated area of the cells in $tile_{i,j}$.

3.4.1.1 Dynamic inflation adjustment

Since our congestion estimation method may over-estimate the routing demand, we need to avoid inflating cells too much. If the total inflation area $TotalInflation$ is larger than a threshold $InflationThreshold$, we will adjust the inflation ratio $InflateRatio$ of each tile in the list $InflationList$ dynamically to avoid over-inflation. This will usually happen at the beginning iterations of the global placement process, since the overflow at the beginning is much larger than that in the later iterations. We will describe this dynamic adjustment as follows.

We will keep a list of tiles called $InflationList$ in which the tiles are sorted by their inflation ratios in a non-decreasing order. Let $tile_{i,j}$ be the first one in $InflationList$. The inflation ratios of all the tiles in $InflationList$ (including $tile_{i,j}$) will be adjusted by a ratio of $1/InflateRatio_{i,j}$. As a result, the inflation ratio of $tile_{i,j}$ becomes 1 instead of $InflateRatio_{i,j} > 1$. Since the sizes of the cells in $tile_{i,j}$ will not be changed (inflation ratio is 1), we will remove $tile_{i,j}$ from $InflationList$. The new area of each cell k in the remaining tiles can then be computed according to equation 3.11 and the $TotalInflation$ can be updated by equation 3.10.

$$NewArea_k = CellArea_k \times (InflateRatio_{i',j'} / InflateRatio_{i,j}) \tag{3.11}$$

This process of removing tiles from $InflationList$ and adjusting the inflation ratios of the remaining tiles will continue until $TotalInflation$ is less than $InflationThreshold$.

Recall that we will alternate between horizontal and vertical cell inflation a few times (six in our implementation) in an iteration. The cell sizes will not be reset until the end of the whole iteration. As described above, there is a parameter called $InflationThreshold$ that limits inflation area for each time of cell inflation. Although the over-inflation can be avoid if $InflationThreshold$ is no more than 1/6 of the chip whitespace[2], and the total inflation area after cell inflation for six times will not exceed 100% of the chip whitespace, we find that allowing a little over-inflation at the beginning iterations of the global placement can help to converge and relieve congestion in the subsequent iterations. For each time of cell inflation, we set $InflationThreshold$ to 30% of the chip whitespace. And in our experiments, the total inflation area per iteration (after cell inflation for six times) does not exceed 110% of the chip whitespace at most. As congestion improves, the inflation area will decrease and the total inflation area per iteration will fall below the chip whitespace after a few iterations.

3.4.2 Cell spreading

After cell inflation, we will spread the cells in those tiles whose cell density is larger than a threshold γ (γ=0.95 in our implementation). A *modified* lookahead

[2] In our implementation, the chip whitespace is a constant. It is calculated before cell inflation.

legalization will be performed to reduce cell density as described as follows.

In our implementation of regular lookahead legalization step [30], a rectangular containing region (referred as expansion region in this paper) will be found for each cell cluster with density larger than γ. In our implementation, the expansion region is found by searching around the bounding box of the cell cluster with uniform distance in all four directions until cell density $\leq \gamma$. The cells will then be distributed evenly within the expansion region, avoiding fixed blocks.

In our modified lookahead legalization step, if we want to reduce the congestion in the horizontal direction, the expansion region will be found by going more in the vertical direction (up and down). In that case, the horizontal demand will be distributed more sparsely in a larger vertical scope. As illustrated in Fig. 3.9, if the inflated cells are spread more in the vertical direction, the horizontal congestion will be alleviated more. Starting with the bounding box of a cell cluster, the ratio of searching expansion region in the vertical and horizontal directions will be 3:1 to resolve horizontal congestion. The vertical congestion is handled similarly but with the ratio of searching expansion region 1:3.

After applying cell inflation and spreading in the horizontal and vertical directions alternately a few times, the routing densities in both directions will be alleviated effectively. All the cells will then be reset back to their original sizes after the last lookahead legalization of an iteration.

3.4.3 Analysis of cell inflation and spreading

In each iteration of our global placement, we will gradually reduce overflow by cell inflation and spreading a number of times (six in our implementation). We will work on the congestion in the horizontal and vertical direction separately by applying cell inflation and spreading in these two directions alternately. In our

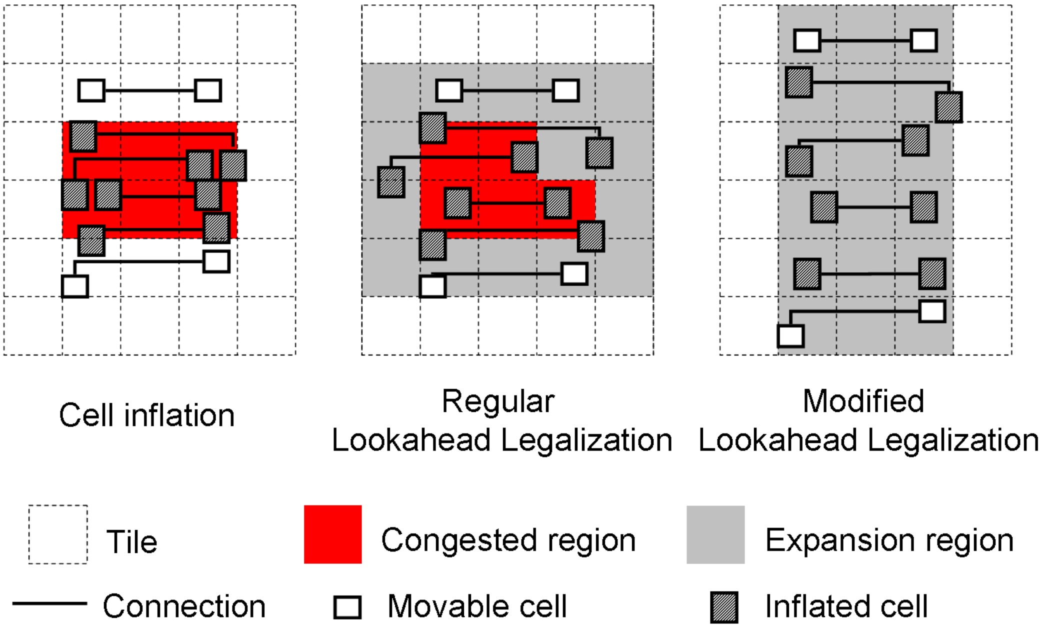


Figure 3.9: An example to show the modified lookahead legalization which is more effective in reducing congestion in one particular direction.

implementation, we will work on the horizontal congestion in the first, third and fifth round and work on the vertical congestion in the second, fourth and sixth round. Fig. 3.10 shows the trajectory of overflow in the horizontal and vertical direction in one iteration. We can see that the horizontal congestion is reduced a lot in the first, third and fifth round while the vertical congestion is not affected too much in those rounds. Similarly, the vertical congestion is improved a lot without affecting the horizontal congestion much in the second, fourth and sixth round.

It is advantageous to reduce congestion in the horizontal and vertical direction separately because the amount of congestion in these two directions is usually different and we can reduce the congestion effectively by processing the two directions independently. It should be mentioned that, after cell inflation and spreading a few times in an iteration, the cells will be reset to their original sizes before the next iteration. We choose not to remember the inflated sizes because the congestion map will be changed from one iteration to another. Without resetting to the

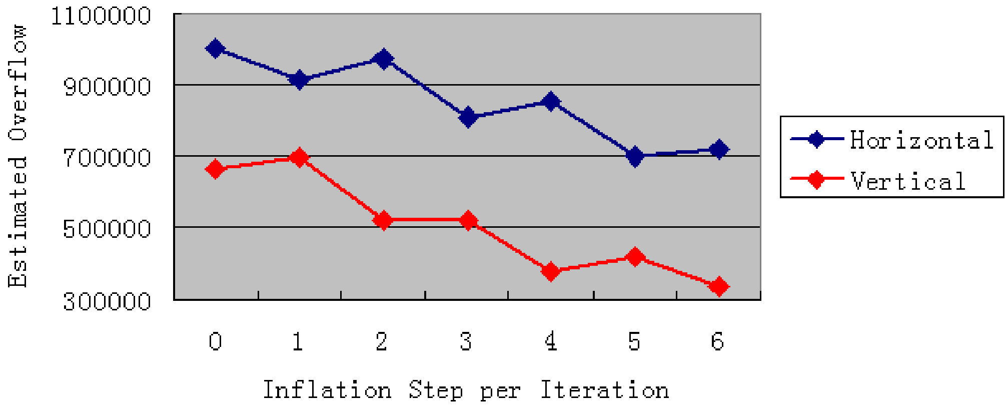


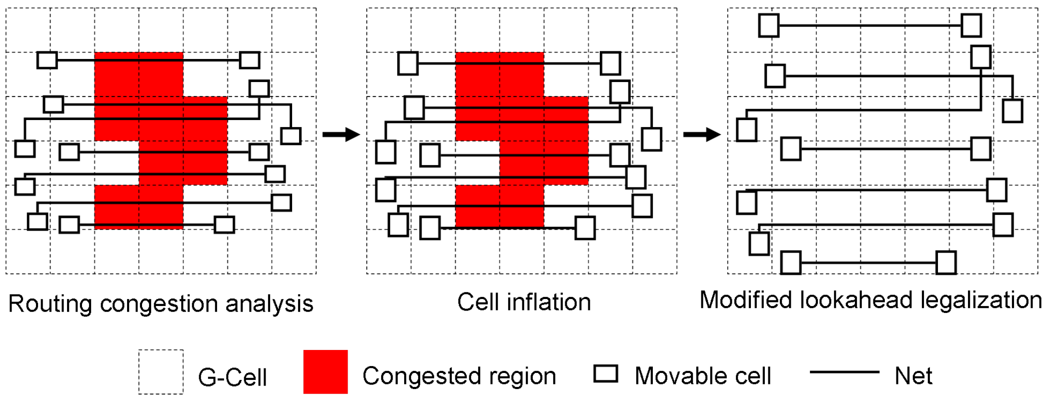
Figure 3.10: Trajectories showing the horizontal and vertical overflow in different steps of a global placement iteration (the 38th iteration of superblue12).

original sizes, the HPWL will get larger and larger while the overflow cannot be reduced quickly as less whitespaces is available for cell inflation. Now, since the cells are always inflated based on the congestion map in the current iteration, the HPWL will not be increased unreasonably and the cells can be inflated effectively to relieve congestion.

3.4.4 Routing path-based cell inflation & spreading

If we use simplified FastRoute with pin density consideration (Section 3.3.2) to do routing analysis, comparing with probabilistic congestion estimations, this lookahead routing estimation not only can produce more accurate congestion map, but also can provide a routing solution for each net.

After routing congestion analysis, we can identify some congested regions where the routing demand is larger than the supply. In the next step, we make use of cell inflation and spreading to reduce the congestion problem. Different from many previous cell inflation approach in which only the cells within the congested regions will be inflated, we propose a routing path-based cell inflation and


Routing congestion analysis

Cell inflation

Modified lookahead legalization

Figure 3.11: An example of routing path-based cell inflation and spreading.

spreading technique to deal with the congestion problem. This new technique is based on the following two reasons.

1. Empirical study shows that a large part of congestion is caused by global nets [38], i.e., the nets with all its connecting cells located outside the congested region. This part of routing congestion cannot be solved by inflating only the cells within the congested region. This problem is more critical when congestion appears on top of macro blockages, which is often the case in modern designs. In such situation, there would be no cell in those congested regions and traditional cell inflation will fail to deal with the problem.

2. Since we make use of a router to perform routing congestion analysis, the routing path of each net in the congestion map is known. We know that, among various options, the router will try to route a net by choosing the least congested path. This kind of information should be utilized in dealing with the congestion problem but was often overlooked.

Consider the horizontal direction in the following example. The key idea of the cell inflation and spreading step can be illustrated in Fig. 3.11. For each decomposed two-pin net, we trace its routing path for a certain distance (at most 10 GCell edges in our implementation) and find the most congested horizontal GCell edge e. If the supply $SupplyH_e$ is less than the demand $DemandH_e$, we will inflate the height of the corresponding cells of this two-pin net by equation (3.12)

$$InflateH_i = InflateH_i \times \frac{DemandH_e}{SupplyH_e}, \tag{3.12}$$

where $InflateH_i$ is the inflated height of cell i. $InflateH_i$ is equal to the original height H_i of cell i before inflation. For a cell associated with several nets, we will inflate it according to the maximum inflation ratio.

After cell inflation, some regions may contain more cells than it can contain. Then, the modified lookahead legalization [18] is performed to roughly legalize the layout. Cell inflation combined with the modified lookahead legalization will spread the cells in the vertical direction. In this case, more routing resources will be provided to the problematic nets that pass through congested regions. This can effectively reduce the congestion problem without a significant change of the layout. For vertical congestion, we inflate the cell width and perform horizontal spreading in the same way. As we can see, this path-based cell inflation technique can relieve the congestion caused by the local, semi-global and more importantly the global nets.

After the horizontal and vertical inflation and spreading, we will perform routing congestion analysis again and this process will be repeated a number of times k ($k = 3$ in our implementation). The rationale behind is that we want to spread the cells in a more careful way to avoid over-spreading. Each time, we spread the cells slightly and perform analysis to see the impact, and the resulting congestion map will be used to guide the next spreading. The inflated height $InflateH_i$ and width $InflateW_i$ will be reset to H_i and W_i before the next iteration starts.

3.4.5 Congested cluster optimization

When congestion and HPWL nearly converge after a number of iterations, we find that there may still be some congested clusters in the layout. Many of these congested clusters have cells with a lot of interconnections [26]. The cells in these clusters need to be spread more sparsely. We show the placement layout and its routing result by NCTUgr [34] of superblue3 in Fig. 3.12(a) and (b) respectively. Fig. 3.12(b) shows the routing hotspots in dark red or violet color. We use white circles to highlight these routing hotspots. Many congested clusters appearing are occupied by a lot of cells (by comparing Fig. 3.12(a) and (b)). If the cells are spread more sparsely, the routing hotspots will be eliminated.

Identification of congested clusters is performed by applying NCTUgr [34] directly. NCTUgr uses the techniques of pattern routing, monotonic routing, maze routing and post routing to relieve congestion. After using these techniques, the remaining congested regions usually contain cells with too many interconnections with each other and the congestion problem cannot be resolved yet.

Adjustments of cell sizes and pseudo-net weight are used after identification of congested clusters. If cell i is in a congested region, we choose to adjust either its height or width with a larger inflation ratio. We budget the adjustment in height by equation (3.13), and the adjustment in width can be similarly computed.

$$H_i = H_i \times \frac{DemandH_{e_1} + DemandH_{e_2}}{SupplyH_{e_1} + SupplyH_{e_2}}, \tag{3.13}$$

where e_1 and e_2 are the edges connecting the GCell containing cell i on the left and right side respectively. The adjusted size will be remembered and used in the following iterations of the global placement. It should be mentioned that the cells will be reset to their original sizes after global placement.

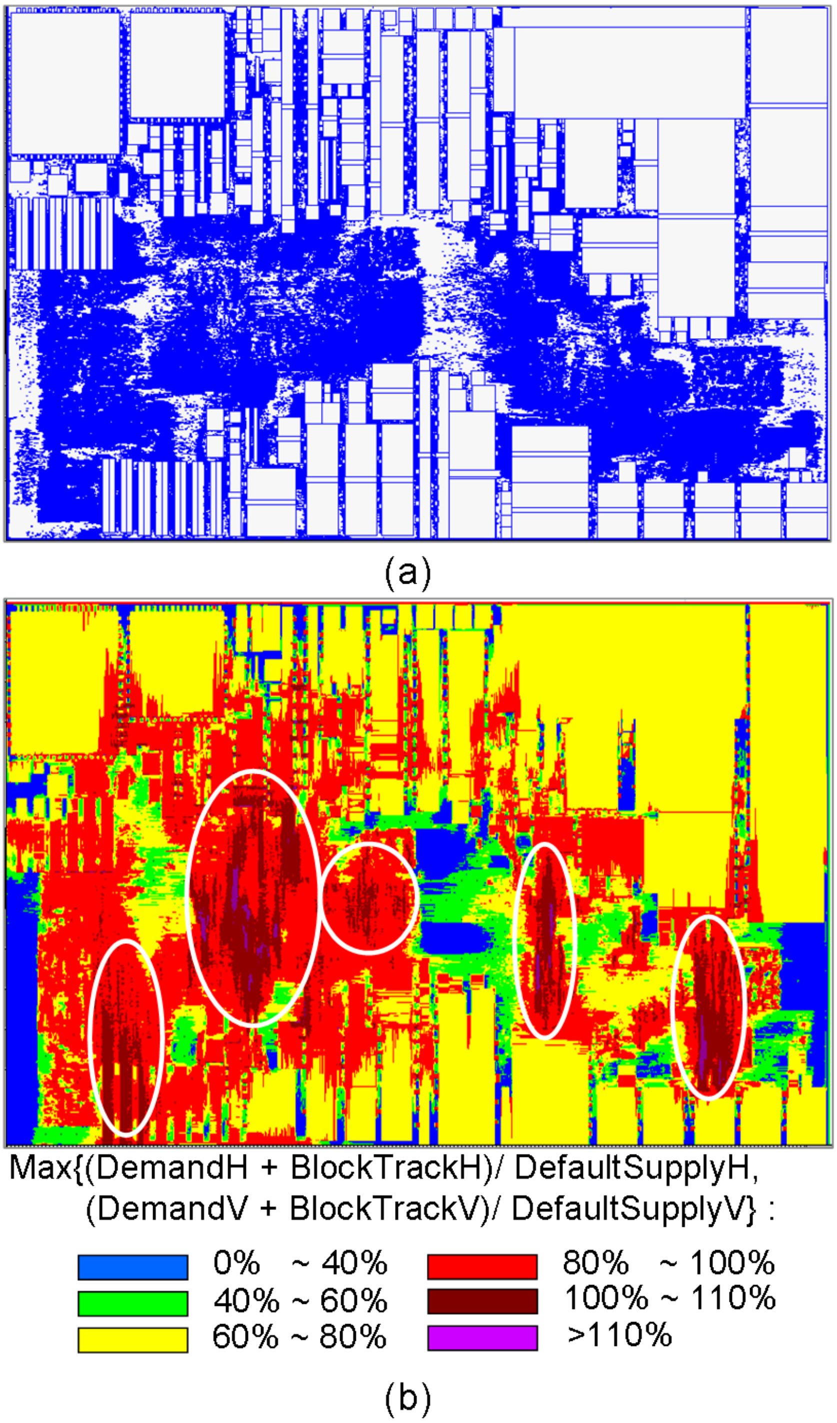


Figure 3.12: The placement layout (a) and its routing result by NCTUgr [34] (b) of superblue3.

Artificial two-pin pseudo-nets are introduced in the lower bound computation to consider the target positions of cells obtained in the upper bound computation.

The pseudo-net weight is computed by equation (3.14).

$$pseudo_i = \alpha/(|lx_i - ux_i| + |ly_i - uy_i|) \tag{3.14}$$

where (lx_i, lx_j) and (ux_i, uy_i) are the coordinates of cell i obtained by the lower and upper bound computation of the previous iteration and $\alpha = 0.01 \times (1+$ iteration number$)$.

After changing cell sizes, HPWL will be increased. In order to avoid worsening HPWL, we perform some additional iterations to reduce HPWL. During these additional iterations, the impact of the upper bound computation is reduced by decreasing the pseudo-net weight in the lower bound computation. After these additional iterations, we will identify congested clusters and adjust cell sizes again. This process of congested cluster optimization will be repeated several times until the congestion is improved obviously. The value of α in the pseudo-net weight (shown in equation (3.14)) is illustrated in Fig. 3.13. At the beginning, α is increased linearly. When we perform the process of congested cluster optimization on the intermediate layouts, α will be adjusted to a lower value and increased linearly again in the subsequence iterations. Since the process of congested cluster optimization is repeated several times, α will be adjusted repeatedly as well.

Fig. 3.14 shows the routing result after applying the congestion cluster optimization. Comparing with Fig. 3.12 (b), the area of the congested regions has been reduced a lot.

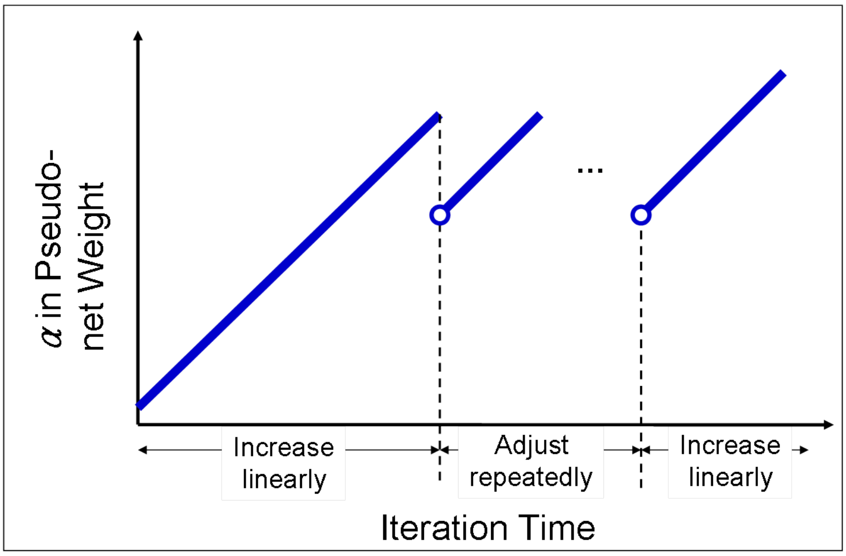


Figure 3.13: The weight of pseudo-net in each iteration of global placement.

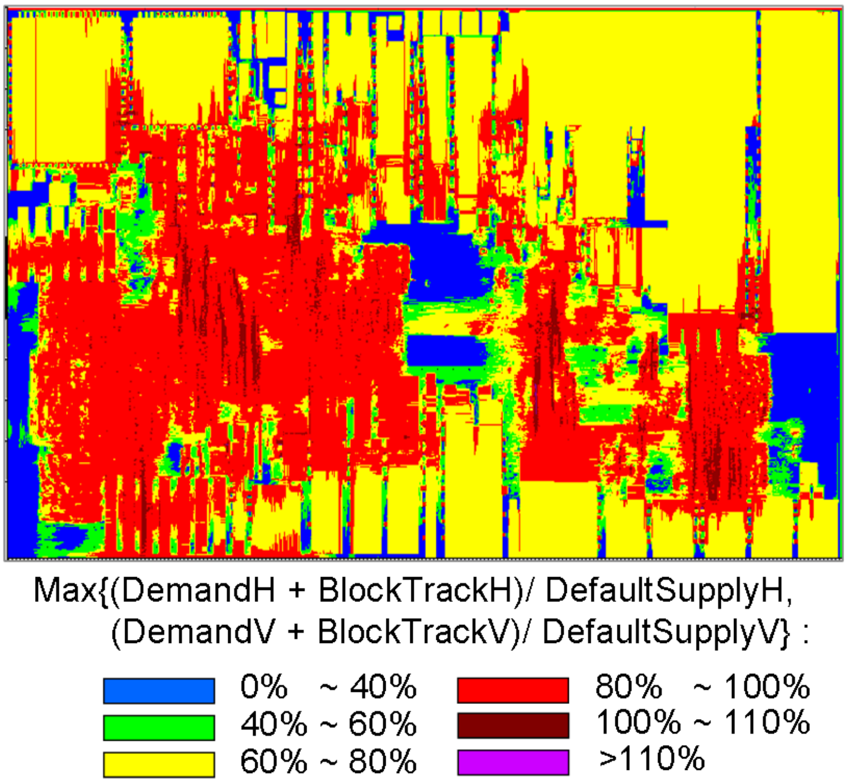


Figure 3.14: After congested cluster optimization, the area of the routing congested regions of superblue3 is much less than the result shown in Fig. 3.12 (b).

CHAPTER 4

Routability-driven legalization and detailed placement

Contents

In this chapter, we study the routability problem in detailed placement. In detailed placement, the most common approach to improve routability is to change the objective of cell swapping to alleviate routing congestion [18][29][21][42][66]. The aim of this chapter is to propose several effective techniques to improve routability in detailed placement of Ripple. The rest of this chapter is organized as follows. In Section 4.2, we review the techniques used in FastPlace-DP [40] which is a traditional placer for HPWL optimization. Section 4.3 describes the techniques used in our legalization and detailed placement in detail.

4.1 Problem formulation

Given a netlist $N = (E,V)$ with nets E and nodes V, the routing resource supplied in upper-metal layer (refer to Section 1.2), and the positions of the movable nodes (cells) obtained by global placement, the legalization process is to remove overlaps between cells, while satisfying design objectives including routing congestion and HPWL. In row-based design, the placement area is partitioned into rows, and the cells and macros must be aligned with the rows after legalization process.

Detailed placement works on the legalized placement to further improve the solution quality. It is more constrained than global placement as it optimizes the objectives by transforming one legal placement solution into another.

4.2 Traditional legalization and detailed placement

We use FastPlace-DP [40] as our basic framework for legalization and detailed placement. In this section, we introduce the methods of HPWL optimization used in FastPlace-DP [40]. The routability techniques in our placer Ripple will be discussed in section 4.3.

4.2.1 HWPL-driven legalization

In our row-based placement, all the macros are fixed. We only need to legalize the standard cells. In legalization, each row is divided into segments based on the overlaps between fixed blockages and the row. A placement segment is the maximal strip in a row not being covered by any blockages. The cells are first moved among segments to satisfy the capacity constraints of the segments. The cells are then legalized within their segments.

When moving cells among segments, we need to determine the target segment for movement. For each cell in a segment, we computer eight scores based on moving the cell to its nearest possible positions in eight neighboring segments. The score is a weighted sum of two parts: (1) the HPWL reduction after movement, and (2) the utilization of the source and target segment. The weight of the second part is higher, since the objective of legalization is mainly to spread cells evenly. If all the scores are negative, the cell will not be moved to any other segment. Otherwise, it will be moved to the segment with the highest score. In each iteration, all the segments with utility above capacity will be handled. The iteration will be repeated until all the segments are within their respective capacities. At last, the cells will be spread to legal positions within each segment.

4.2.2 HWPL-driven detailed placement

The detailed placer FastPlace-DP [40] used in Ripple works only on the standard-cells in a legalized row-based placement or a placement in which the macros have been fixed. There are three major techniques to further reduce the HPWL of a legalized layout: (1) cell swapping, (2) local reordering, and (3) single segment clustering. During cell swapping, global swapping and vertical swapping are performed. The flow of the detailed placement is summarized in Fig. 4.1.

4.2.2.1 Cell swapping

For each cell i, a swapping window for cell i can be calculated for wirelength optimization. The idea behind cell swapping is to swap i with a cell j or a space s in the swapping window of i.

For global swapping, the swapping window is called the optimal region. The optimal region of a cell is determined based on the median idea in the work [16].

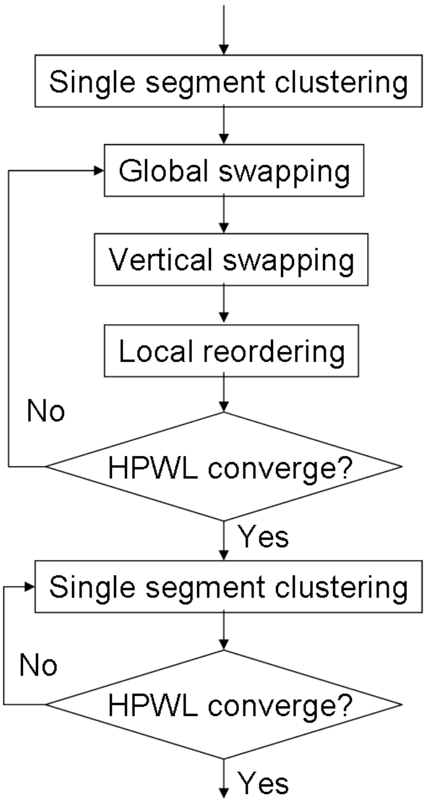


Figure 4.1: Detailed placement flow in FastPlace-DP.

When calculating the optimal region of cell i, all its associated nets N_i and their bounding boxes are found first. It should be mentioned that cell i is excluded from the nets when computing their bounding boxes. For each net $p \in N_i$, its bounding box is noted as $x_l[p]$,$x_r[p]$,$y_b[p]$,$y_r[p]$ (the left, right, bottom, and up boundaries). According to the bounding boxes, two series can be obtained: ($x_l[1]$,$x_r[1]$,$x_l[2]$,$x_r[2]$,...,) and ($y_b[1]$,$y_u[1]$,$y_b[2]$,$y_u[2]$,...,). The optimal position of cell i is given by (x_{opt},y_{opt}), where x_{opt} and y_{opt} are the medians of the x and y series respectively. Usually, the number of elements in these two serious are even, and the optimal position is a rectangular region rather than a point. In some cases, the optimal region may degrade to a point/line when the two medians of the x and/or y series carry the same value.

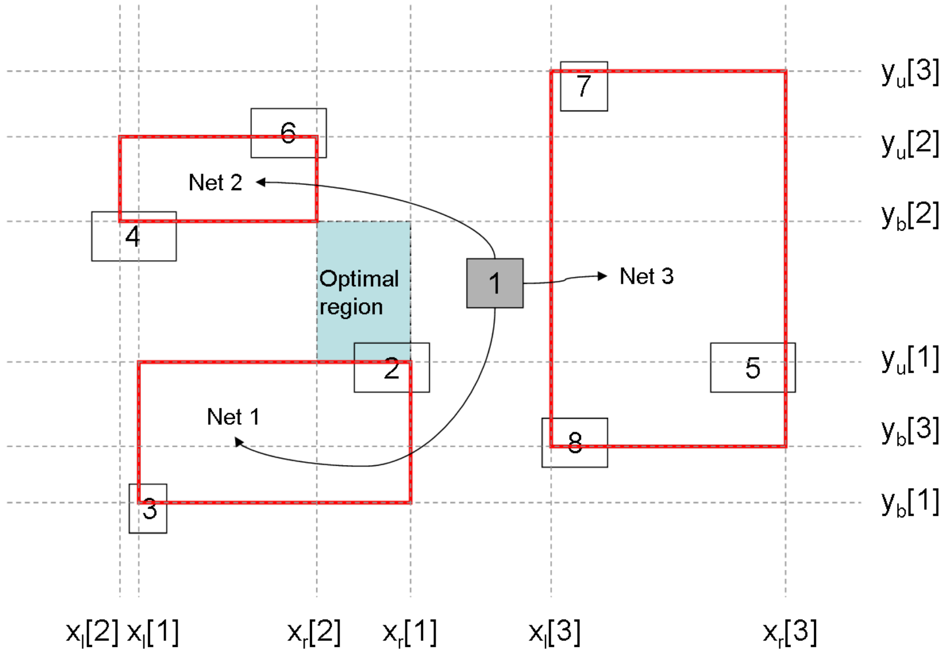


Figure 4.2: Optimal region of cell 1.

Fig. 4.2 shows an example of computing the optimal region for cell 1. There are three nets connecting with cell 1: $Net1$, $Net2$ and $Net3$. The bolded rectangles in red color are the bounding boxes of these nets excluding cell 1. The x and y series are $(x_l[1], x_r[1], x_l[2], x_r[2], x_l[3], x_r[3])$ and $(y_b[1], y_u[1], y_b[2], y_u[2], y_b[3], y_u[3])$. The region in blue color is the optimal region of cell 1.

After finding the optimal region for cell i, we will try to move i into this region. Before moving, we have to determine the cell/space being swapped with i. If there is any overlap created by swapping, the neighboring cells have to be shifted. A benefit for the swapping and shifting effect on wirelength is introduced in [40]. For each cell i, we try to swap it with every cell j and space s in its optimal region. The benefit is measured for each tentative swapping. At last, the j or s with the best benefit will be picked and swapped with i. If the best benefit is less than zero, we do not perform swapping for cell i.

Vertical swapping is similar to the global swapping. It is used to increase the possibility of a good swap and reduce the vertical wirelength locally [40]. Vertical swapping only moves a cell up or down by one row. After calculating cell *i*'s optimal region, we will check whether the optimal region is above or below *i*'s current location. Then, a few cells in the segment above/below cell *i* will become swapping candidates. The same benefit of swapping will be measured as in the process of global swapping.

It is observed that comparing with only applying global swapping, interleaving vertical swapping with global swapping results in a much faster decrease in wirelength. The reason is that vertical swapping is not as greedy as global swapping, and it is more flexible in moving the cells. Moreover, it may aid the global swapping [40].

4.2.2.2 Local reordering

With vertical swapping fixing local vertical errors, local reordering is used to fix local horizontal errors. For any n consecutive cells within a segment, all possible ordering of the cells are tried, and we pick the order with the best wirelength. When reordering the n cells, we consider them as a group. The left and right boundary of the group is the left and right boundary of the first and last cell respectively in the original order. For each order, the cells are distributed evenly within the left and right boundaries. In our implementation, n is equal to 3 to tradeoff the runtime and the HPWL optimization. [1]

[1] We also tried $n = 4$ without obvious improvement.

4.2.2.3 Single segment clustering

Given a segment S in the placement region with n cells whose left-to-right order is: $c_1, c_2, ..., c_n$, assuming that all the cells not in S are fixed, the clustering problem is to find a non-overlapping placement for the segment S so that the HPWL is minimized. FastPlace-DP [40] proposed an efficient algorithm to find the optimal solution.

During cell clustering in segment, a cluster is a cell or a group of cells abutting together in the original order. Clustering is to merge two clusters as a new cluster whose width is the sum of the widths of the original clusters. The wirelength function of the x-coordinate of a cluster is a convex piecewise linear function $W(x)$ when all other objects are fixed. The slopes for the linear pieces are $..., -3, -2, -1, 0, 1, 2, 3, ...$, and it is unnecessary that all integer slopes exist. The part with slope 0 is the optimal region in the x-direction for the cluster. The points where the function changes slope are called bounds. These bounds are the left and right boundaries of the bounding boxes of the nets associated with the cluster (excluding the cluster itself).

When finding the optimal region for a cluster in segment S, the positions of all the other cells and macros except the cells in S are fixed. Although the bounds of the cluster cannot be determined if it has any connection with the cells in S, we fix the order of processing the cells in S so that the optimality of the solution will not be affected. Since the cells' coordinate in segment S is not fixed, when we compute the bounding box of net N associated with the cluster, if a cell c' is in S and left/right to the cluster, we will assume c' is at the left/right end of the segment S. Although the real coordinate of c' is not used, it will not affect the optimality because the left-right order is maintained.

During clustering, every cluster will try to be put at the center of its optimal region. If there is overlap between two clusters, they will be merged as a new cluster and processed as a single cluster at any later stage. The new cluster will be put at the center of its optimal region. This clustering process will continue until all the clusters are placed without any overlap. If the optimal region boundary is out of the segment range, it will be assign to the closest segment boundary. The proof of optimality is given in [40].

4.3 Routability-aware legalization & detailed placement

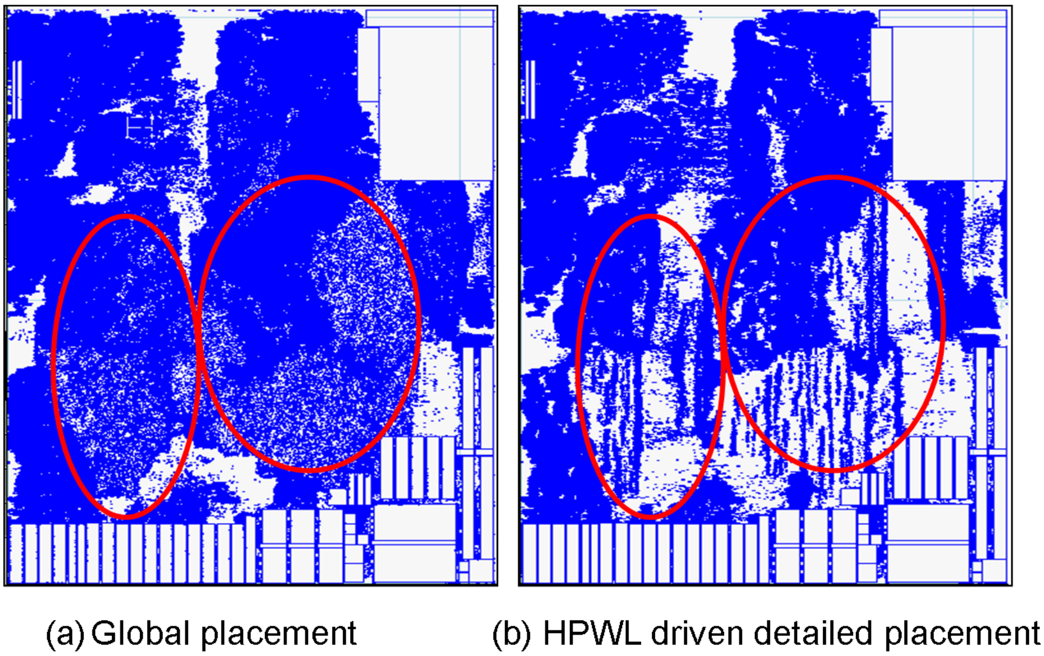
We use FastPlace-DP [40] as our basic framework for legalization and detailed placement. However, FastPlace-DP is HPWL-driven which will increase congestion and reduce routability. As shown in Fig. 4.3(a), after global placement, the cells are located quite sparsely in the potentially over-congested regions. However, after legalization and detailed placement, the cells are moved closer to each other (Fig. 4.3(b)) and the overflow of the next global routing step will very likely be increased significantly. We have studied how to modify the HPWL-driven approach in legalization and detailed routing to trade off congestion and HPWL.

4.3.1 Congestion-driven legalization

Legalization is usually thought to have less impact on the placement result since we just legalize the global placement result. However, we find that it can worsen congestion in the horizontal direction quite significantly.

In FastPlace-DP, several steps are applied to do legalization:

- Step 1: Move cells out of macro blocks.



(a) Global placement (b) HPWL driven detailed placement

Figure 4.3: Superblue12 before and after a HPWL-driven detailed placement. Regions highlighted in red circles have high routing congestion.

- Step 2: Spread cells in their corresponding segments under the constraint of not exceeding the segment capacity.[2]
- Step 3: Legalize cells within each segment. The overlaps between cells are removed within each segment.

Among these steps, Step 3 is the most possible one to worsen the horizontal congestion. Here, we use FastRoute [39] to verify the impact of Step 3 on overflow in the horizontal direction. In order to check the overflow fast, we use simplified FastRoute, which is FastRoute without the monotonic routing and maze routing step. Although we do not use the full process of FastRoute, the result is still quite illustrative. Table 4.1 shows that Step 3 will worsen the congestion in the horizontal direction by more than two times. The influence on the vertical congestion is small according to our experiments. Take superblue12 as an example, although

[2] Segments are strips in a row separated by fixed nodes.

Table 4.1: Horizontal overflow after each step of legalization. Overflow is generated by the simplified FastRoute

Benchmark	After Global	After Legalization		
		After Step 2	After Step 3	After Modified Step 3
superblue1	25675	30002	67727	38653
superblue2	60252	61210	88562	72756
superblue4	89944	93345	145179	107464
superblue5	102826	102829	190777	113411
superblue10	129836	133966	197685	148745
superblue12	201645	198634	553444	216982
superblue15	94734	106739	334733	129848
superblue18	87818	88047	128310	95874
Avg.	1.00	1.03	2.15	1.17

its horizontal overflow after legalization is worsen a lot (as shown in table 4.1), its vertical overflow before and after legalization are 128559 and 126571 respectively.

We find that before Step 3, the total area of the cells in a segment does not exceed the segment capacity. Since the cells have overlaps with each other, Step 3 will try to relocate the cells in a segment to remove overlaps. If the length of a segment is very long, the cells will be legalized by spreading a long distance from their original positions. The large displacement from the global placement result after legalization has worsened the congestion. Fig. 4.4 shows an example to illustrate this problem. The region in the circle has a lot of overlaps. After being legalized by Step 3, the cells are spread by a long distance away from their original x-coordinate. Therefore, the routing demand and overflow in the horizontal direction is changed a lot.

In order to solve this problem, we simply limit the maximum length of a segment. If the length is shorter, the capacity of each segment will be less, and Step 2 will try to move cells to their neighborhood segments (including the segments in the upper and bottom rows) with less displacement. This method can easily solve

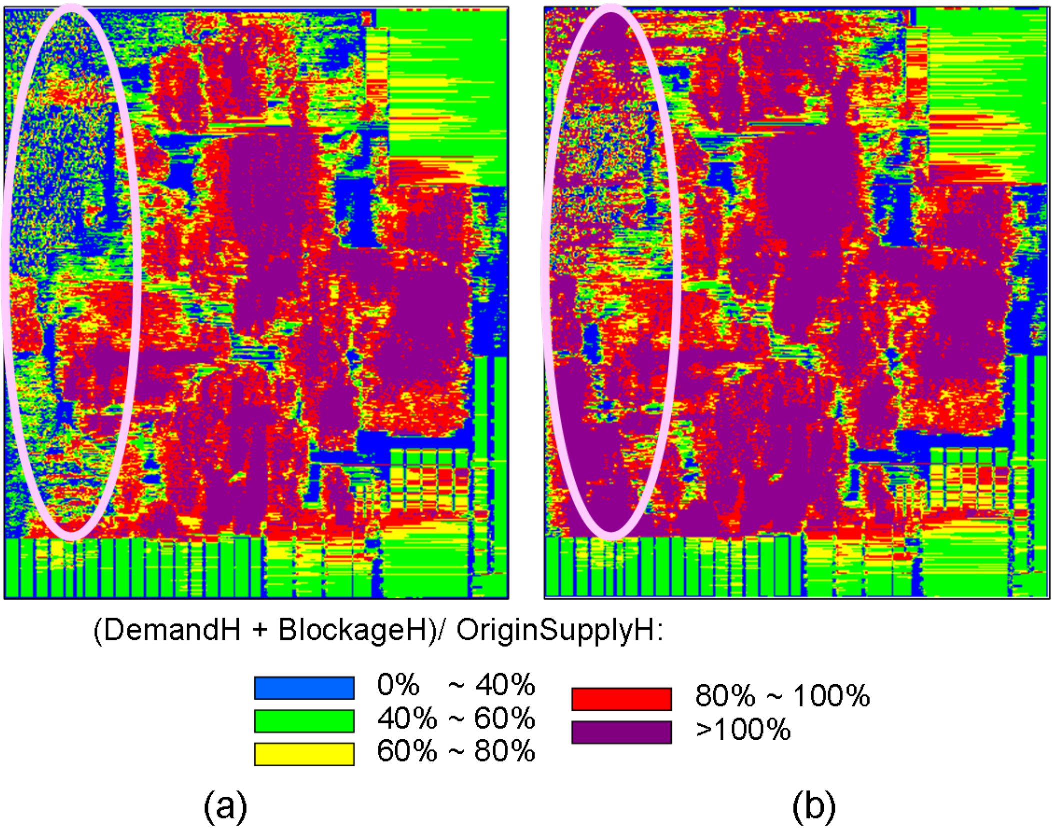


Figure 4.4: An example to show the effect of legalization on the horizontal congestion (superblue12). (a) The horizontal congestion map before legalization. (b) The horizontal congestion map after legalization. Both congestion maps are generated by the simplified FastRoute.

the problem discussed above. In our implementation, the maximum length of a segment is set to $\max(3 \times maxCellWidth, 0.02 \times chipWidth)$, where $maxCellWidth$ is the maximum width of a movable cell and $chipWidth$ is the width of a chip. If the length of a segment is larger than the maximum length, we will divide it into uniform small segments. Note that we use the same setting for all the benchmarks in our experiments. Table 4.1 shows the congestion after Step 3 with limit on the length of the segments. We can see that the overflow is not worsen as much.

4.3.2 Congestion-driven detailed placement

Many previous works try to modify the cell swapping process to improve overflow in detailed placement [18][29][21]. However, we find that the impact of cell swapping on congestion is not the most significant one. We have studied carefully the steps in FastPlace-DP:

- Clustering step: Recursively cluster cells that are neighboring to each other in each segment to reduce HPWL.
- Swapping step: Swap between two cells, or swap a cell to an empty location (including global cell swapping and vertical cell swapping).
- Reordering step: Re-order a small number of neighboring cells in each segment.

These three steps, with each one containing many iterations, will be performed one by one. For example, in the clustering step, clustering will be performed recursively for many times until the HPWL is not improved. Table 4.2 shows the overflow right before detailed placement and the relative change in overflow after each step in detailed placement. These overflows are obtained by the simplified

Table 4.2: Total overflow (horizontal and vertical) caused by each step in detailed placement. Overflow is generated by the simplified FastRoute.

Benchmark	Legalization	Detailed		
		Cluster	Swap	Reorder
superblue1	155112	+22280	+21280	-24494
superblue2	1079291	+323009	+85292	-50418
superblue4	166270	+5653	+27150	-34501
superblue5	299240	+27793	+45387	-45938
superblue10	496319	+83518	+67290	-40856
superblue12	331669	+514536	+313315	-312190
superblue15	178893	-16326	+25758	-53542
superblue18	152383	+5773	+44225	-39543
Avg.	1.0	+33.8%	+22.0%	-21.0%

FastRoute as discussed in section 4.3.1. We find that the clustering step will increase congestion most.

The clustering step can on the other hand improve HPWL a lot, so it is not good to remove it from detailed placement. If we check the overflow caused by clustering in each segment of the chip, the runtime will become too long. In order to minimize running time and not to worsen congestion, we choose to check the overflow after each clustering or swapping iteration using the simplified FastRoute. As explained above, there are many iterations in each of those clustering and swapping phases. After each iteration in clustering and swapping, the simplified FastRoute will be invoked to check whether the overflow is larger than a certain threshold (we use the overflow after legalization as the threshold). If the overflow exceeds the threshold, we will proceed to the next phase. We find that at the beginning iterations, both the overflow and HPWL can be reduced (HPWL is reduced more). By checking the overflow after each iteration, we can choose a right time to stop a process and proceed to the next one.

CHAPTER 5

Simultaneous routing and placement (SRP) for congestion refinement

Contents

In this chapter, an effective simultaneous routing and placement refinement tool called SRP is proposed for routability improvement.

In post-placement process, there are some previous works considering congestion refinement. The most recent ones, CROP [66] and CRISP [44], address routability in post-placement process directly. After obtaining the congestion map, CROP [66] adjusts the boundary of each GCell and shifts the modules according to the new GCell boundary. CRISP [44] applies cell inflation and spreading to allocate cells more sparsely in the congested regions.

Global router overlays a regular grid (GCell) on the chip, and constructs a global routing on the grid. In a global routing grid graph, each GCell is represented by a node, and there are global routing edges to connect adjacent GCells. If the usage of routing tracks on an edge *e* is above *e*'s capacity, this edge *e* is considered as over-congested. Given a globally routed layout, the congestion of a region can be caused by three types of nets: (1) local nets connecting cells within the region, (2) semi-global nets coming in/out of the region, and (3) global nets passing through the region without connecting any cell of the region. Many previous works [66][44] can directly solve the overflow from local nets by relocating cells in congested GCells. However, as we know , the overflow caused by global or semi-global nets is usually more than that caused by local nets. Therefore, we should consider congestion caused by all types of nets in our routability refinement method. Since many placers have already used whitespace allocation method during placement, we choose to do cell relocation directly during our refinement process. Besides, other refinement tools can also integrate with SRP to further improve the congestion.

The remainder of this paper is organized as follows. In Section 4.2, we give an overview of our post-placement process. Section 4.3 describes the details of our simultaneous relocating and rerouting algorithm. Section 4.4 gives further discussion on SRP. Experimental results will be given in Section 4.5.

5.1 Problem formulation

Given a netlist $N = (E, V)$ with nets E and nodes V, the routing resource supplied in upper-metal layer (refer to Section 1.2), the positions of the nodes obtained by detailed placement, and the routing results of all nets given by global router (e.g. NCTUgr [34]), the refinement tool SRP is to replace movable cells, rip-up and reroute nets with the goal of minimizing the total routing overflow on GEdges. To show the effectiveness of our work, we also uses NCTUgr [34] to evaluate the routability of the placement solutions produced by SRP (discussed in Section 6.3).

5.2 Overview of SRP

SRP is independent of the placer and global router being used. Based on a given placement layout and its global routing result, we try to relocate cells and reroute nets to reduce overflow. Three major steps are used in our tool. First, we identify and rip-up problematic cells by considering congestion caused by all types of nets mentioned above. Different from many previous works that only focus on reducing the overflow caused by local nets, our method may relocate a cell even when its routing path obtained by the global router has just run across congested regions. Second, we search for new location for a cell by multi-source propagation. Unlike other works that only search the surrounding region around the cell's original location, we will find a new location for the problematic cell by searching around the GCells passed through by the routing paths of the associated nets of the cell. Third, we will connect the problematic cell from its new location to its associated nets by a multi-subnet maze routing algorithm.

Fig. 5.1 shows the overall flow of SRP. The input is a placement layout and its global routing result. Before relocating and rerouting, we need to obtain the routing capacity and the routing demand according to the global routing result. If the routing demand of a region is larger than its routing capacity, this region is a congested region. During cell relocation and net rerouting, every problematic cell that has some connections going across some congested regions will be identified and ripped up first. Since not only the cells in the congested region, but also the cells that have connections going across congested regions will be identified and relocated, the congestion cause by local nets, semi-global nets and global nets are all handled. Then, the problematic cell will be relocated and its associated nets will be rerouted. If the total overflow is improved, the cell will stay at its new location, and the congestion map will be updated according to the rerouted paths; otherwise, the cell will go back to its original position. In each iteration, all the problematic cells will be processed one after another. The whole post-placement process will be finished if there is no more obvious reduction in overflow. To trade off between congestion improvement and runtime, the number of iterations is 3 in our implementation.

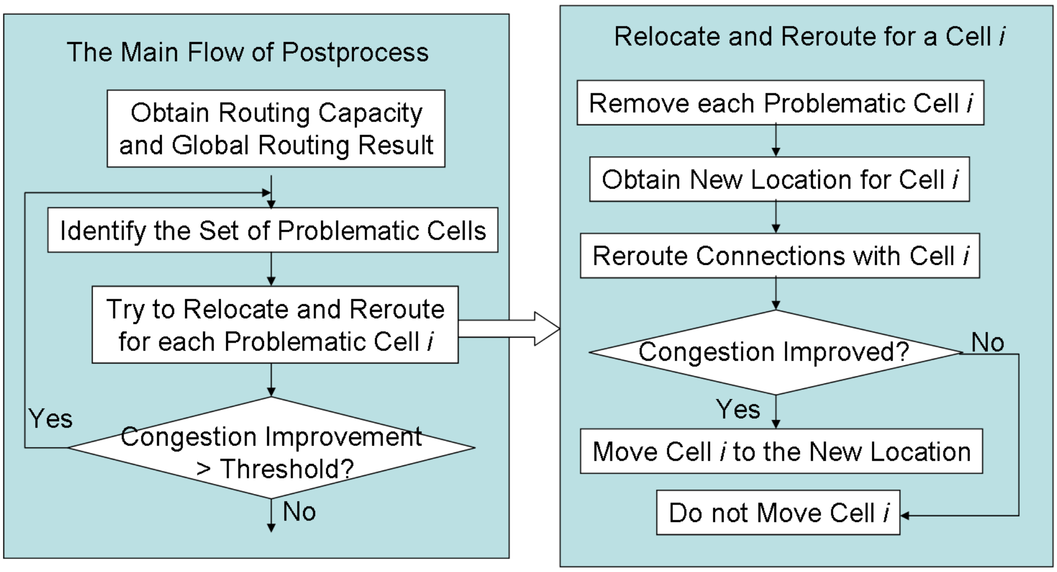

The Main Flow of Postprocess
Obtain Routing Capacity and Global Routing Result
Identify the Set of Problematic Cells
Try to Relocate and Reroute for each Problematic Cell *i*
Yes
Congestion Improvement > Threshold?
No
Relocate and Reroute for a Cell *i*
Remove each Problematic Cell *i*
Obtain New Location for Cell *i*
Reroute Connections with Cell *i*
Congestion Improved?
No
Yes
Move Cell *i* to the New Location
Do not Move Cell *i*

Figure 5.1: The whole flow of post-process.

5.3 Simultaneous cell relocation & net rerouting

The chip is divided uniformly into $m \times n$ GCells. Each pair of adjacent GCells have a edge connecting them. If the pins of a net is in different GCells, there is a routing path going through edges to connect these pins. Each edge e has a capacity, which is the total number of available tracks for wires to go through. If the total number of used tracks (routing demand) is larger than the capacity of the edge, there will be overflow on this edge.

5.3.1 Identify problematic cells

After obtaining the routing result, the cells with connections going across congested regions should be relocated to reduce the overflow of those regions. Different from previous works which only relocate the cells in the congested regions, we also identify the cells whose connections have passed through congested regions. By doing this, not only the overflow caused by local nets but also the overflow caused by semi-global nets and global nets can be reduced.

Given the routing result of a net, we will first identify the congested GCells passed through by the routing path of this net. If a congested GCell has at least one pin of this net, the cell of this pin will be marked as a problematic cell. Otherwise, we will follow the routing path from this congested GCell until reaching other GCells that contain some pins of this net, and then the cells of these pins will be marked as problematic cells. After identifying all the problematic cells, we will relocate them to reduce overflow.

Take net A in Fig. 5.2 as an example. Net A has three cells: cell 1, cell 2 and cell 3. The routing path (in grey color) of net A is shown in Fig. 5.2 (a). This path passed through a congested routing edge (in red color) connecting $tile(1,2)$ and $tile(2,2)$.

Since $tile(1,2)$ has a pin of cell 1, cell 1 will be marked as a problematic cell. Since $tile(2,2)$ doesn't contain any pin of net A, we will follow the routing path until we reach $tile(4,3)$ that contains a pin of cell 2. Then we stop propagating, and mark cell 2 as a problematic cell.

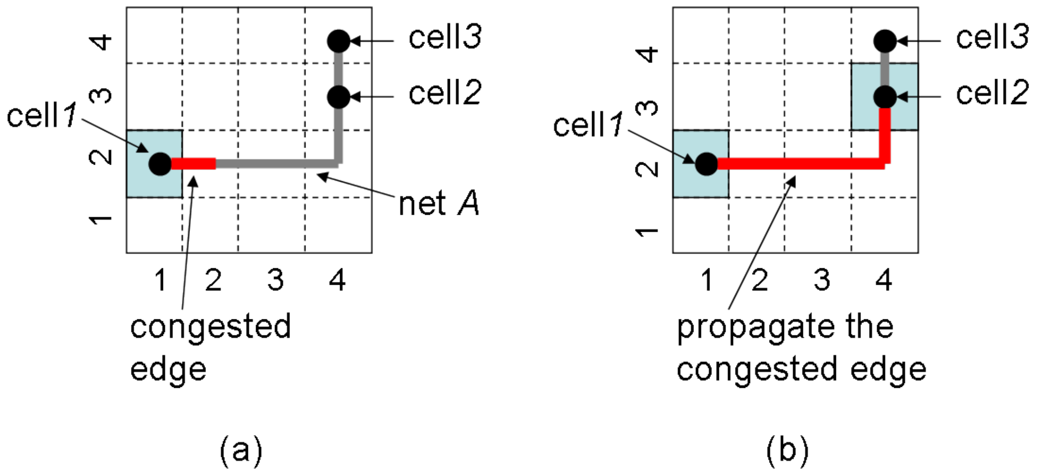


Figure 5.2: (a)The routing path of net A passes through a congested routing edge connecting $tile(1,2)$ and $tile(2,2)$ (b) We propagate from $tile(1,2)$ and $tile(2,2)$ until reaching any pins of net A. Cell 1 and cell 2 are marked as problematic cells.

5.3.2 Remove problematic cells

Before problematic cell relocation and net rerouting, we first need to remove the problematic cell and rip up the routing paths connecting this cell. There are three cases need to be considered (shown in Fig. 5.3) when we delete the routing path connecting with a problematic cell:

Case 1: In Fig. 5.3 (a), cell 1 is a problematic cell. Since $tile(1,1)$ contains cell 1 which only has degree one, we will remove the cell directly, and delete the routing path connecting to $tile(1,1)$. After deleting the path from $tile(1,1)$ to $tile(2,1)$, the degree of $tile(2,1)$ becomes one. Since there is no pin in $tile(2,1)$, we can delete the path from $tile(2,1)$ to $tile(3,1)$ as well. This deleting process

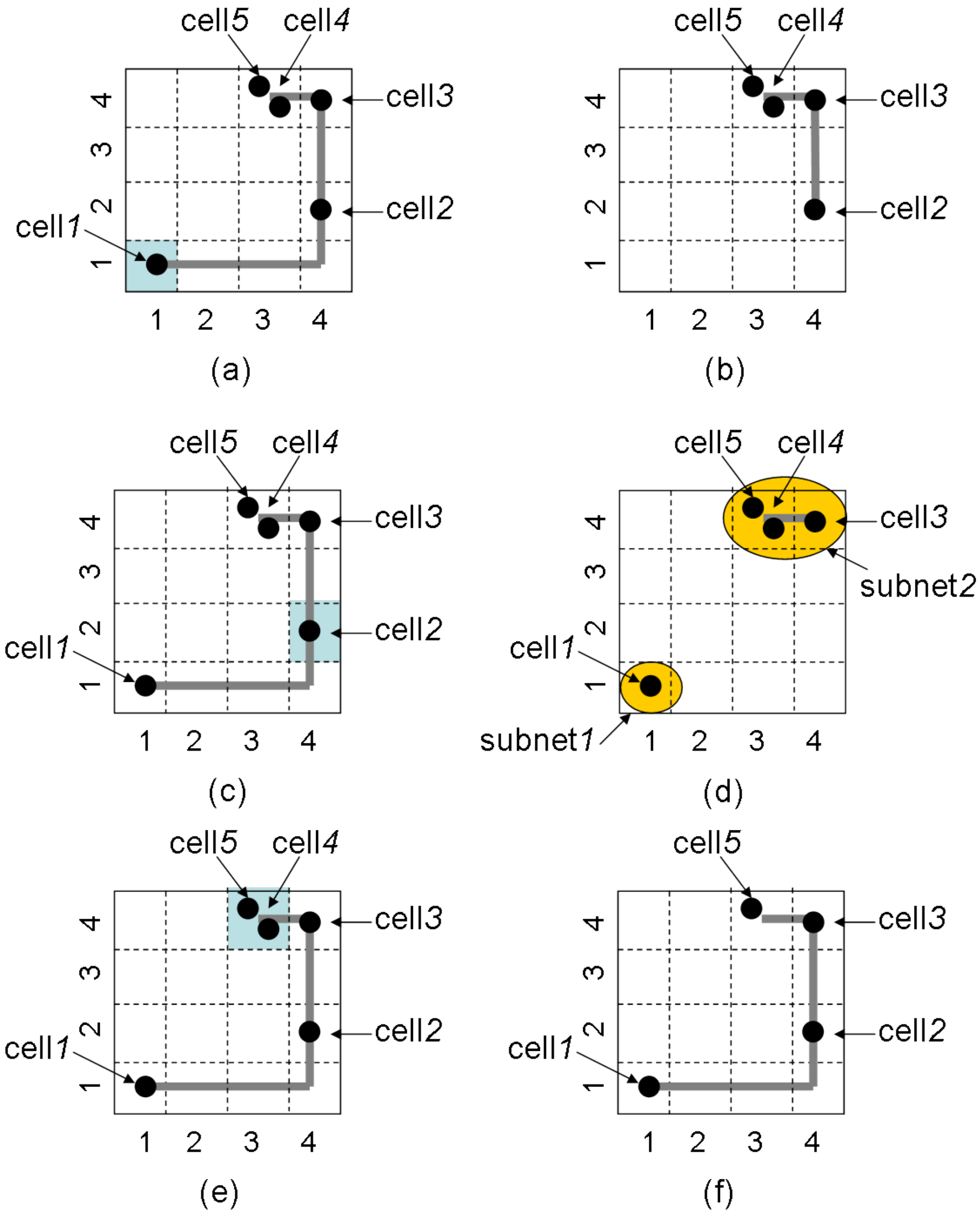


Figure 5.3: (a) $Tile(1,1)$ contains a problematic cell 1 (b) The remaining path and cells after removing cell 1 (c) $Tile(4,2)$ contains a problematic cell 2, and the degree of this tile is two (d) After removing cell 2, the remaining path has two subnets (e) $Tile(3,4)$ has two cells, and cell 4 is the current problematic cell (f) The remaining path and cells after removing cell 4.

will continue until reaching a GCell that contains a pin or a steiner point of this net.

The remaining routing path of this net after removing cell 1 is shown in Fig. 5.3

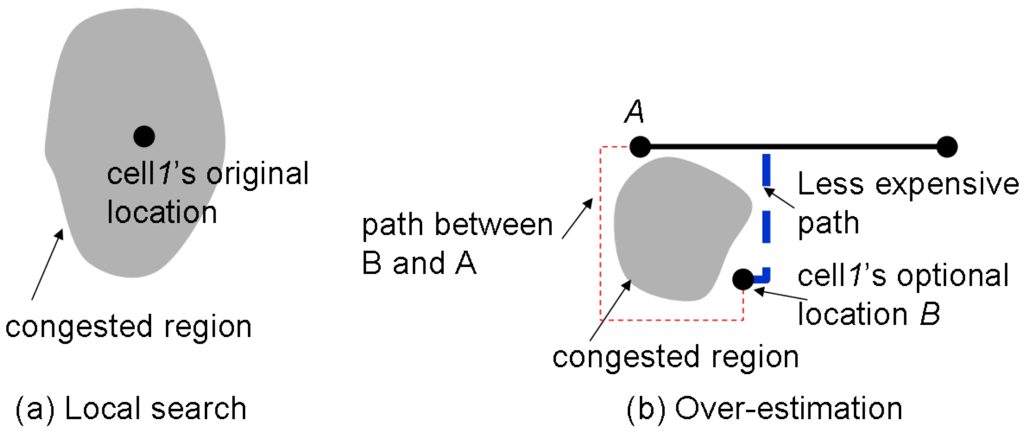


Figure 5.4: Potential problems of some previous approaches.

(b).

Case 2: In Fig. 5.3 (c), cell 2 is a problematic cell. $Tile(4,2)$ contains cell 2 which has degree more than one ($1 \leq degree \leq 4$). In order to reduce the overflow caused by the paths connected to cell 2, we choose to delete the path from $tile(4,2)$ to $tile(4,4)$ and the path from $tile(4,2)$ to $tile(1,1)$. After deleting these paths connected to cell 2, the net becomes two subnets (shown in Fig. 5.3 (d)). We will discuss how to connect cell 2 and these subnets together after searching a new position for cell 2 in section 5.3.3 and section 5.3.4.

Case 3: In Fig. 5.3 (e), cell 4 is a problematic cell. However, $tile(3,4)$ contains cell 4 and cell 5 which are both on this net. After removing cell 4, we cannot delete the routing path connecting to this GCell, since cell 5 is still in this GCell. Fig. 5.3 (f) shows the remaining path and cells of this net after removing cell 4.

5.3.3 Searching new location

During global placement and detailed placement, most previous works [42][32][45][64] find the new locations of the problematic cells by searching the surrounding GCells of the cells' original locations. Besides, before evaluating the cost of each optional GCell, the multi-pin nets are first decomposed into two-pin subnets. These two scenarios are illustrated in Fig. 5.4 to show the potential problems of their strategies.

Algorithm 1 Multi-Source Propagation

```
for net $i \in$ associateNet($curCell$) do
    for $tile_a \in$ path(net $i$) do
        $cost_i(tile_a) = 0$
        enqueue($tile_a$, $Q_i$)
    end for
end for
while $numOptionTile < optionThreshold$ do
    for net $i \in$ associateNet($curCell$) do
        $tile_a$=dequeueMin($Q_i$)
        if $!isVisitTile(tile_a, i)$ then
            $isVisitTile(tile_a, i)$=true
            $numNetReach(tile_a) + = 1$
        end if
        if $numNetReach(tile_a) ==$ |associateNet($curCell$)| then
            $numOptionTile + = 1$
        end if
        for $tile_b$ adjacent with $tile_a$ & $!isVisitTile(tile_b, i)$ do
            $cost_i(tile_b) = cost_i(tile_a) + cost_i(tile_a, tile_b)$
            enqueue($tile_b$, $Q_i$)
        end for
    end for
end while
```

Local search: Consider the scenario in Fig. 5.4 (a). The problematic cell 1's location and its surrounding regions are all congested. If we only consider the surrounding regions as optional new locations, either the cell will not be moved (because of no overflow improvement) or the cell has to be moved several times until it reaches a good position out of the congested region.

Over-estimation: Consider the scenario in Fig. 5.4 (b). After decomposing a net into two-pin subnets, problematic cell 1 is connected with the pin at location A directly. When we evaluate the cost of an optional location B, we will try to connect

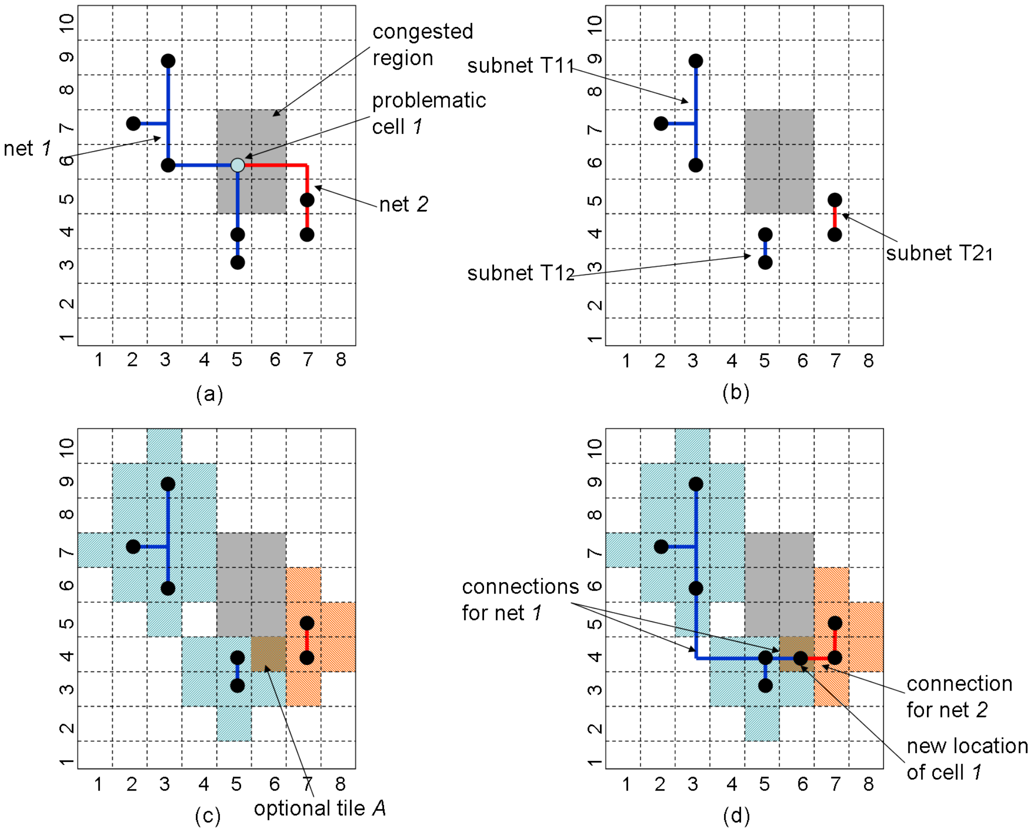


Figure 5.5: (a) The problematic cell 1 has two associated nets: net 1 and net 2. (b) After removing cell 1 and its connections from net 1 and net 2, the remaining path is partitioned into two subnets: $T1_1$ and $T1_2$; while net 2 becomes subnet $T2_1$. (c) Obtain an optional location for cell 1 by using multi-source propagation method from both net 1 and net 2. (d) Find the shortest path to connect the new location and the subnets of together.

B with A. However, besides point A, B can actually be connected to any other GCell passed by the net (thickened blue line), and the cost may become smaller. Therefore, the cost of locating cell 1 in B may be over-estimated.

As we can see in these two scenarios, the remaining routing path of the net is not utilized in an effective way when searching for a new location for the problematic cell. Not only the positions around the cell's original location, but also the positions surrounding any GCells passed through by the remaining routing path of the net should be considered. Besides, when connecting the problematic cell to its associated net, we only need to find a path between its new location to any GCells passed through by the remaining routing path of the net.

Aware of this problem, we propose a multi-source propagation method to obtain the new location. After removing a problematic cell from its associated nets, the GCells passed through by the remaining routing paths of its nets will be treated as sources. Then, we apply the multi-source propagation method on each associated net i of the problematic cell. The sources on each net i will propagate simultaneously until a GCell is reached by all the associated nets. As shown in Fig. 5.5 (a), the problematic cell 1 is associated with two nets: net 1 and net 2. After ripping up cell 1 and deleting the paths at these two nets connecting with it, net 1 is partitioned into two subnets $T1_1$ and $T1_2$; while net 2 becomes subnet $T2_1$ (shown in Fig. 5.5 (b)). All the GCells passed through by $T1_1$ and $T1_2$ are considered as sources of net 1. Similarly, the GCells passed through by $T2_1$ are the sources of net 2. We will propagate from these two sets of sources until a GCell A is reached from these two nets. Then GCell A will be an optional position for cell 1.

Algorithm 1 shows our multi-source propagation method. Given a problematic cell $curCell$, for each associated net i of $curCell$ ($i \in associateNet(curCell)$), a priority queue Q_i is maintained. Initially, the sources of net i are enqueued into Q_i, with the cost equal to zero (line 1-6). From each Q_i, we dequeue a tile $tile_a$ with the minimum cost (line 9-13). Once a tile $tile_a$ has been visited by all the associated nets, $tile_a$ can be an optional location for the problematic cell $curCell$ (line 14-16). After dequeuing $tile_a$ from Q_i, we will propagate from $tile_a$ to its adjacent tile $tile_b$ (line 17-20). The cost of visiting $tile_b$ is the cost of $tile_a$ added by the cost $tile_a$ to $tile_b$. Both the overflow and the wirelength of the edge connecting from $tile_a$ to $tile_b$ are considered when calculating the cost from $tile_a$ to $tile_b$. The cost function is shown in equation (5.1). The loop in line 8-21 is similar to the Dijkstra's algorithm. The loop in line 7-22 will be repeated until the number of optional locations is equal to a threshold $optionThreshold \geq 1$, (e.g., $optionThreshold$=2 in our experiments).

$$\begin{aligned} cost_i(tile_b) &= cost_i(tile_a) + cost_i(tile_a, tile_b) \\ cost_i(tile_a, tile_b) &= of(tile_a, tile_b) + wl(tile_a, tile_b) \end{aligned} \tag{5.1}$$

$$cost(tile_k) = \sum_{net_i \in N} cost_i(tile_k) \tag{5.2}$$

We use equation (5.2) to evaluate each optional location $tile_k$. The tile with the minimum cost will be chosen as a new location for the cell. The method to generate new paths between the cell's new location and its associated subnets will be illustrated in section 5.3.4.

5.3.4 Connections to new location

After finding a new location for the problematic cell, we can trace back according to the propagation process (section 5.3.3) to find the shortest path from this new location to the cell's associated nets. However, there are two reasons not to use this method.

First, not all the subnets of a net can be connected together. An associated net may be partitioned into more than one subnets after ripping up the problematic cell. Since tracing back can only find a path from the new location to one GCell passed through by one subnet. Therefore, only one of the subnets can be connected. As shown in Fig. 5.6 (a), the new location of cell 1 can only be connected with subnet 1 when tracing back, while subnet 2 cannot be connected.

Second, the path may not be the shortest path when tracing back. If the problematic cell has more than one associated net, the path obtained by tracing back may not be the shortest one since the costs on the edges may change. As shown in

Algorithm 2 Multi-Subnets Maze Routing

```
while $numSubnet$ >1 do
    for $tile_a \in$ path(subnet 1) do
        $cost(tile_a) = 0$
        enqueue($tile_a$, $Q$);
    end for
    while $|Q|$ >0 do
        $tile_a$=dequeueMin($Q$)
        if $!isVisitTile(tile_a, i)$ then
            $isVisitTile(tile_a, i)$=true
            if isSink($tile_a$) then
                $i$=subnetID($tile_a$)
                break
            else
                for $tile_b$ adjacent $tile_a$ & $!isVisitTile(tile_b, i)$ do
                    $cost(tile_b) = cost(tile_a) + cost(tile_a, tile_b)$
                    enqueue($tile_b$, $Q$)
                end for
            end if
        end if
    end while
    path(subnet 1)=path(subnet 1) $\cup$ path(subnet $i$) $\cup$ path($tile_a$, subnet 1)
    updateRoutingDemand(path($tile_a$, subnet 1))
    $numSubnet -= 1$
end while
```

Fig. 5.6 (b), the problematic cell 4 has three associated nets: net 1, net 2, and net 3. After removing cell 4, the remaining cells of these three nets are cell 1, cell 2, and cell 3 respectively. Since both net 1 and net 2 will pass through $tile(1,3)$ and $tile(2,3)$ after tracing back, the overflow along these two paths will be increased comparing the values used during the propagation process. Therefore, the paths obtained by tracing back the propagation process may not be the shortest ones for net 1 and net 2 any more.

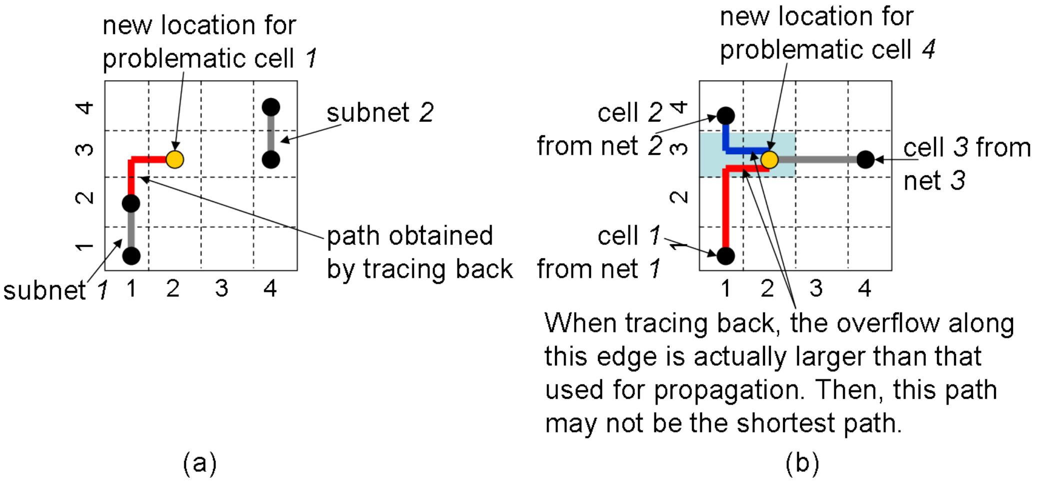

new location for problematic cell 1
subnet 2
path obtained by tracing back
subnet 1
1 2 3 4
1 2 3 4
(a)
new location for problematic cell 4
cell 2 from net 2
cell 3 from net 3
cell 1 from net 1
1 2 3 4
2 3 4
When tracing back, the overflow along this edge is actually larger than that used for propagation. Then, this path may not be the shortest path.
(b)

Figure 5.6: (a) The path obtained by tracing back can only connect the problematic cell 1 to one of its subnets (subnet 1), and subnet 2 is not connected. (b) Problematic cell 4 has three associated nets. Since both net 1 and net 2 will pass through $tile(1,3)$ and $tile(2,3)$ after tracing back, the overflow on these two paths will be larger comparing with the value in the propagation process. Therefore, the paths obtained by tracing back the propagation may not be the shortest paths for net 1 and net 2 respectively.

Therefore, we will connect the new location to each associated subnet one by one, and update the overflow of the routing edges after obtaining the shortest path for each net.

In order to find the shortest path to connect all the subnets of a net, we use multi-subnets maze routing. If the new location of the problematic cell is not passed through by any subnets, this new location will be seen as a subnet as well. The algorithm of multi-subnet maze routing is shown in Algorithm 2. We first take all the GCells on one subnet (e.g., subnet 1) as the sources (line 2-5), and all the GCells on other subnets as sinks. We propagate from the sources by using the cost function in equation (5.1) (line 6-17). Once we reach a sink $tile_a$ (line 8-10), we will get the subnet ID i in $tile_a$. Then subnets 1 and subnet i will be connected as a new subnet by the $path(tile_a, subnet1)$. The overflow of all the routing edges passed through by $path(tile_a, subnet1)$ will be updated. All the GCells on this new subnet will become sources for the next propagation process (line 18-19). This propagation process will repeat until all the subnets are connected.

Fig. 5.5 (d) shows the paths connecting between cell 1's new location and the other subnets.

5.4 Further discussions

Most placers have already used whitespace allocation method as a major technique to reduce overflow during global placement or detailed placement. If we use similar technique in the refinement process, the improvement in congestion may not be large. Therefore, we choose to relocate and reroute problematic cell directly.

The number of cells which can be moved to reduce congestion successfully is much smaller than the number of problematic cells. The major reason is that even a new location and a new routing path are found, the overflow caused by the new routing path is hard to be smaller than that of the original one. In our implementation, we change the relocation order of the problematic cells. We sort the problematic cells by the total overflow value of their routing paths in a non-increasing order. The cells with larger overflow values will be relocated and rerouted earlier. Besides, in order to reduce runtime, we ignore the problematic cells whose probability of reducing overflow is too small. For example, these include cells whose routing path overflow value is less than a threshold $thr1$, and the cells whose minimum distance between its original location and the overflow GCell its routing paths passing through is larger than a threshold $thr2$ (e.g., $thr1$ = 2, $thr2$ = 100 in our experiments).

We also tried to move multiple cells together. However, we find that the congestion map is inaccurate when many cell locations and routing paths are updated simultaneously. It is very difficult to determine how to move many cells and reroute many nets simultaneously. Actually, in our implementation, the overflow improvement of moving multiple cells together is even smaller than that of moving cells one by one.

CHAPTER 6

Experimental results

Contents

In this chapter, we first introduce the benchmarks released in contests ISPD 2011 [54], DAC 2012 [52], and ICCAD 2012 [55]. The experimental results of detailed placement and the refinement tool SRP are then analyzed. At last, the results of Ripple are discussed in detail.

6.1 Benchmarks for routability placement

Three sets of benchmarks released in contests ISPD 2011 [54], DAC 2012 [52] and ICCAD 2012 [55] are shown in Table 6.1, Table 6.2, and Table 6.3 respectively. All of them are real industrial ASIC designs from IBM. These benchmarks can be used

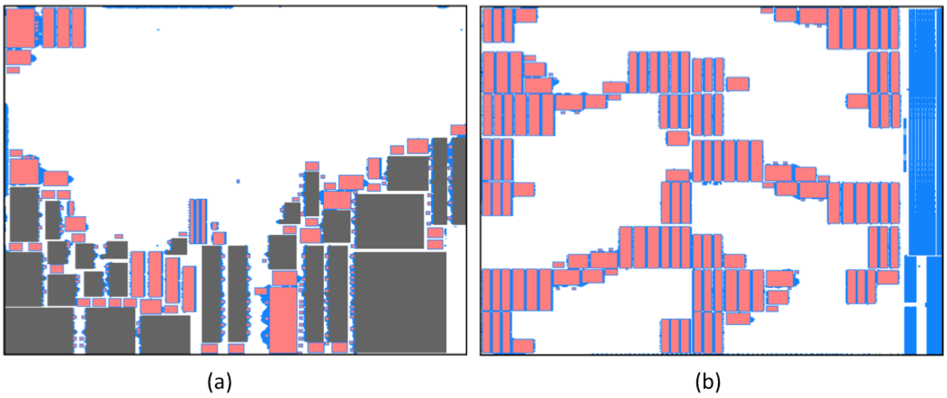


Figure 6.1: (a) The floorplans of fixed terminal nodes in superblue18. (b)The floorplans of fixed terminal nodes in superblue15. The light-red shaded boxes with blue out-line represent the rectangular fixed nodes and the dark-gray shaded boxes represent the non-rectangular fixed nodes in the design [54].

to perform both placement and global routing. Moreover, they are representative of today's designs with numerous placement blockages, more metal layers, varying metal width and spacing across layers, etc.

These benchmarks also demonstrate the varying characteristics and associated challenges of modern ASIC floorplans. For example, all the fixed nodes in superblue18 (shown in Fig. 6.1 (a)) are pushed to the periphery of the placement region, which gives the placement tool a large amount of whitespace in and around the center of the placement region. On the other hand, the fixed nodes in superblue15 (shown in Fig. 6.1 (b)) fragment the placement region into multiple subregions. In addition, the tall and thin fixed nodes in most of the designs create "alleys". Alleys in the placement region can lead to significant congestion if a net gets split across the corresponding fixed nodes.

To consider design hierarchy during physical synthesis, the set of benchmarks in ICCAD 2012 contest contain the design hierarchy information. It is hoped that placement algorithms can potentially use the design hierarchy information to minimize wire length as well as routing congestion and to improve runtime.

Table 6.1: Benchmark in ISPD 2011 contest.

Design	Total Nodes	Movable Nodes	Terminal Nodes	Terminal _NI	Total Nets	Total Pins	Util. (%)	Den. (%)
sb1	847441	765102	52627	29712	822744	2861188	69	35
sb2	1014029	921273	59312	33444	990899	3228345	76	28
sb4	600220	521466	40550	38204	567607	1884008	70	44
sb5	772457	677416	74365	20676	786999	2500306	77	37
sb10	1129144	914921	153595	60628	1085737	3665711	75	35
sb12	1293433	1278084	8953	6396	1293436	4774069	56	44
sb15	1123963	829614	252053	42296	1080409	3816680	73	60
sb18	483452	442405	25063	15984	468918	1864306	67	47

Util.(%) is the area ratio between all nodes and the chip.

Den.(%) is the area ratio between all movable nodes and the whitespace of the chip.

Table 6.2: Benchmark in DAC 2012 contest.

Design	Total Nodes	Movable Nodes	Terminal Nodes	Terminal _NI	Total Nets	Total Pins	Util. (%)	Den. (%)
sb2	1014029	921273	59312	33444	990899	3228345	76	28
sb3	919911	833370	55033	31508	898001	3110509	73	42
sb6	1014209	919093	65316	29800	1006629	3401199	73	43
sb7	1364958	1271887	66995	26076	1340418	4935083	76	58
sb9	846678	789064	37574	20040	833808	2898853	73	47
sb11	954686	859771	67303	27612	935731	3071940	79	40
sb12	1293433	1278084	8953	6396	1293436	4774069	56	44
sb14	634555	567840	44743	21972	619815	2049691	72	50
sb16	698741	680450	419	17872	697458	2280931	69	46
sb19	522775	506097	286	16392	511685	1714351	78	49

The benchmarks superblue2 and superblue12 are the same with that of ISPD 2011 contest.

Table 6.3: Benchmark in ICCAD 2012 contest.

Design	Total Nodes	Movable Nodes	Terminal Nodes	Terminal _NI	Total Nets	Total Pins	Util. (%)	Den. (%)
sb1	847441	765102	52627	29712	822744	2861188	69	35
sb3	919911	833370	55033	31508	898001	3110509	73	42
sb4	600220	521466	40550	38204	567607	1884008	70	44
sb5	772457	677416	74365	20676	786999	2500306	77	37
sb7	1364958	1271887	66995	26076	1340418	4935083	76	58
sb10	1202665	1045874	96251	60540	1158784	3894138	70	32
sb16	698741	680450	419	17872	697458	2280931	69	46
sb18	483452	442405	25063	15984	468918	1864306	67	47

The benchmarks superblue3, superblue7 and supeblue16 are the same with that of DAC 2012 contest.

The benchmarks superblue1 and superblue10 and superblue18 are the same with that of ISPD 2011 contest.

Table 6.4: Comparison between the HPWL-driven detailed placement and the congestion-driven detailed placement. Overflow is generated by the simplified FastRoute.

Benchmark	HPWL-driven		Congestion-driven	
	HPWL	Overflow	HPWL	Overflow
sb1	266643392	174178	269995648	157781
sb2	618041152	1437174	631629376	1133897
sb4	216831264	164572	217536384	156085
sb5	337018464	326482	340494368	289346
sb10	568484544	606271	575723136	500300
sb12	347745536	847330	371608672	308700
sb15	311213824	134783	311498176	134606
sb18	169159472	162838	170792336	143331
Avg.	1.0	1.0	+1.9%	-73.3%

6.2 Experimental result of detailed placement

We use the benchmarks in ISPD 2011 contest to show the effectiveness of our congestion driven detailed placement (discussed in Chapter 4). Table 6.4 shows that our congestion-driven placer can reduce the overflow obtained by the original detailed placer by 73.3% while increasing the HPWL by only 1.9% only. Our congestion-aware detailed placement is simple and efficient enough to trade off between routability and wirelength.

We also use other cell swapping strategy to deal with routability during detailed placement. But the routing overflow is worse and the runtime is much longer. This is because when performing cell swapping, the runtime of computing routing congestion is much longer than that of computing the HPWL. Besides, the routing congestion analysis is not totally the same with the evaluation metric using NCUTgr. In addition, many overflows are generated in clustering step as well. Therefore, only focusing on cell swapping step as many previous placers cannot effectively solve the routability problem.

6.3 Experimental result of SRP

We use the top four placer results in DAC 2012 Contest to test SRP (discussed in Chapter 5). The input placement results are obtained by the top four placers NTUplacer [21], Ripple [18], SimPLR [29] and mpl12 [6] in the routability-driven placement DAC Contest 2012 [52]. The input global routing results are given by NCTUgr [34] which is machine independent.

After using SRP, we use NCTUgr [34] to route the output layouts. We show the total normalized overflow of all metal layers in Table 6.5. The total overflow after using SRP is improved by 12.35%, 8.39%, 75.01%, 34.57% for the placers NTUplacer [21], Ripple [18], SimPLR [29] and mpl12 [6] respectively. The runtime of SRP is shown in Table 6.5. If we use the runtimes of these four placers in the DAC 2012 Contest [52] as reference, the runtime of SRP takes only 8.42% of the total runtime of the placement process on average.

As shown in Table 6.6, the routed wirelength by NCTUgr after using SRP is not affected obviously. Besides, the increased HPWL after using SRP is always very small compared with the input placement.

6.4 Experimental result of Ripple

6.4.1 Study of the basic framework

In our basic framework shown in Fig. 3.1, we use probabilistic estimation to do congestion analysis in global placement (Section 3.3.1). Cells are inflated in congested region using a dynamic inflation adjustment method (Section 3.4.1.1). After global placement, the congestion-driven legalization and detailed placement are processed (Chapter 4). The refinement tool SRP is not used in this basic frame-

Table 6.5: Overflow comparison and runtime of SRP

	NTUplacer			Ripple			SimPLR			mpl12		
	Input	Post		Input	Post		Input	Post		Input	Post	
Benchmark	OF	OF	CPU(s)	OF	OF	CPU(s)	OF	OF	CPU(s)	OF	OF	CPU(s)
sb2	4527	4320	454	51038	47451	1444	114562	90674	1935	248259	185867	1651
sb3	11744	10182	346	39068	36295	974	16191	13689	515	34903	28891	485
sb6	4723	4215	319	7011	6087	467	2753	2506	280	4189	3241	300
sb7	3013	2650	168	13604	13239	381	299288	123573	671	19031	14199	215
sb9	4893	4648	127	13592	12479	342	8389	7420	207	18812	15380	281
sb11	2386	2176	107	2884	2686	200	21617	15953	171	13224	11341	231
sb12	77	59	81	1975	1718	365	2878	2524	298	105472	67374	939
sb14	410	400	45	1750	1592	66	5479	4074	155	2666	2373	77
sb16	7410	6359	105	3831	2850	144	43027	32677	348	5498	4308	145
sb19	1606	1297	47	4352	3939	148	3908	2949	91	21382	18850	100
Avg.	+12.35%	1.00	179.9	+8.39%	1.00	453.1	+75.01%	1.00	467.1	+34.57%	1.00	442.4

Our implementation is written in C++ and compiled with g++ 4.1.2. All the benchmarks are run on an Intel Core 2 Duo Linux workstation with 2.8GHz and 4 GB RAM, using one CPU core.

Table 6.6: Routed wirelength and HPWL comparison

	NTUplacer				Ripple				SimPLR				mpl12			
	RoutedWL		HPWL		RoutedWL		HPWL		RoutedWL		HPWL		RoutedWL		HPWL	
Bench mark	Input (×E6)	Post (×E6)	Input (×E8)	Post (×E8)	Input (×E6)	Post (×E6)	Input (×E8)	Post (×E8)	Input (×E6)	Post (×E6)	Input (×E8)	Post (×E8)	Input (×E6)	Post (×E6)	Input (×E8)	Post (×E8)
sb2	17.89	17.89	6.11	6.12	19.33	19.34	6.51	6.52	18.77	18.77	6.29	6.31	21.20	21.22	7.11	7.13
sb3	10.90	10.90	3.27	3.28	11.66	11.66	3.34	3.35	12.57	12.56	3.63	3.63	12.32	12.32	3.64	3.65
sb6	10.45	10.45	3.30	3.31	10.77	10.77	3.35	3.35	11.18	11.18	3.50	3.50	12.22	12.22	3.87	3.87
sb7	12.74	12.74	3.91	3.91	14.09	14.09	4.23	4.23	14.34	14.29	4.29	4.29	14.38	14.38	4.47	4.48
sb9	7.66	7.66	2.37	2.37	8.44	8.44	2.58	2.58	8.26	8.26	2.54	2.54	9.01	9.01	2.78	2.78
sb11	10.32	10.32	3.42	3.42	10.71	10.70	3.56	3.56	10.55	10.55	3.47	3.47	12.70	12.69	4.23	4.23
sb12	10.98	10.98	3.12	3.12	11.78	11.77	3.33	3.33	12.19	12.19	3.55	3.55	10.85	10.86	3.24	3.26
sb14	6.93	6.93	2.25	2.25	7.09	7.09	2.27	2.27	7.47	7.47	2.38	2.39	8.08	8.08	2.58	2.58
sb16	7.90	7.90	2.62	2.62	7.78	7.78	2.59	2.59	8.13	8.15	2.67	2.68	8.68	8.68	2.89	2.89
sb19	4.68	4.68	1.50	1.50	4.91	4.91	1.59	1.59	4.94	4.94	1.58	1.58	5.36	5.36	1.74	1.74
Avg.	-.002%	1.00	-.04%	1.00	+.002%	1.00	-.05%	1.00	+.03%	1.00	-.10%	1.00	-.05%	1.00	-.15%	1.00

work. The benchmarks in the ISPD 2011 contest are used to illustrate the efficiency of this framework.

The global placement phase mainly consists of five parts: Conjugate Gradient descent (CG), lookahead legalization, congestion estimation, cell inflation and spreading. About 42.9% , 28.5% and 25.4% of the total runtime of the global placement phase is spent on congestion estimation, CG and lookahead legalization respectively. The processes of cell inflation take only 3.2% of the total runtime of the global placement phase.

Fig. 6.2 shows the lower and upper bounds for HPWL, the inflation area and the sum of congestion estimations in both the horizontal and the vertical directions at the end of each iteration for the benchmark superblue18. Since cell inflation is used after the 3rd iteration, the upper-bound HPWL is dramatically increased at the 3rd iteration, and then begins to decrease. We do not want to inflate cells at the beginning, since the overflow is usually too large by our estimation at this stage. This strategy is the same throughout all the benchmarks. The congestion value is reduced greatly in the first 25 iterations, and then the improvement slows down. When the congestion is relieved iteratively, the inflation area also reduces gradually.

Table 6.7 shows the runtime for the global and detailed placement phase in seconds. Most of the runtime is spent on the global placement phase.

6.4.1.1 Comparison with top contestants in ISPD 2011 contest

Table 6.8 lists the overflow and HPWL of the placements obtained by this Ripple basic framework (RippleBasic), and the top results from the ISPD 2011 Contest participants (ISPD2011) [1] and SimPLR [29]. In order to show the efficiency of the dynamic inflation adjustment step discussed in section 3.4.1.1, we also use a

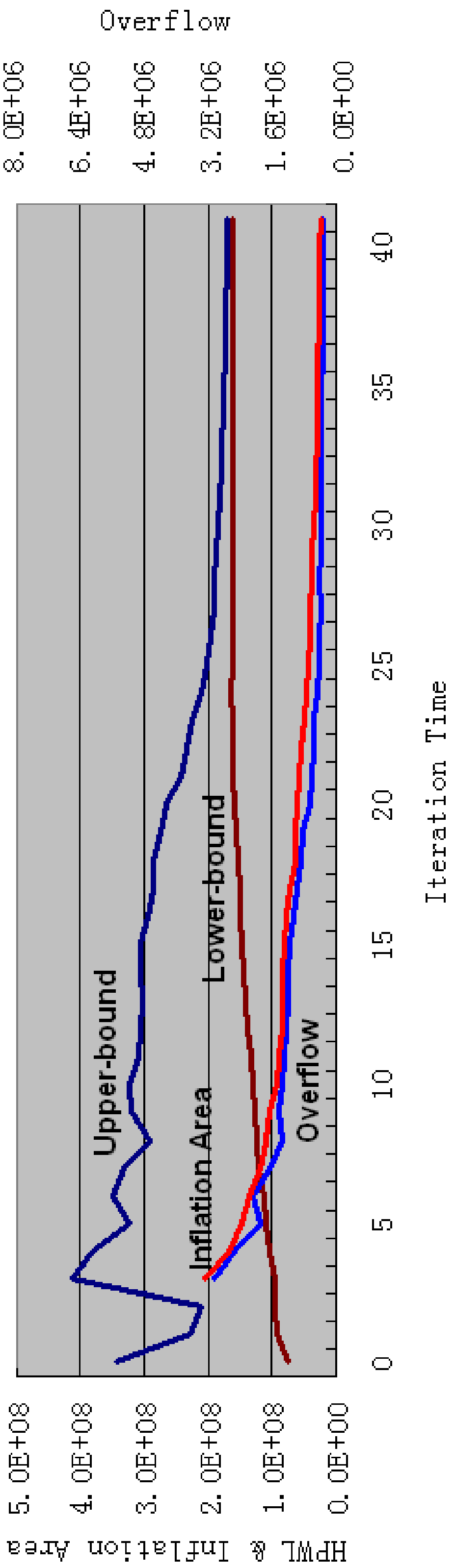
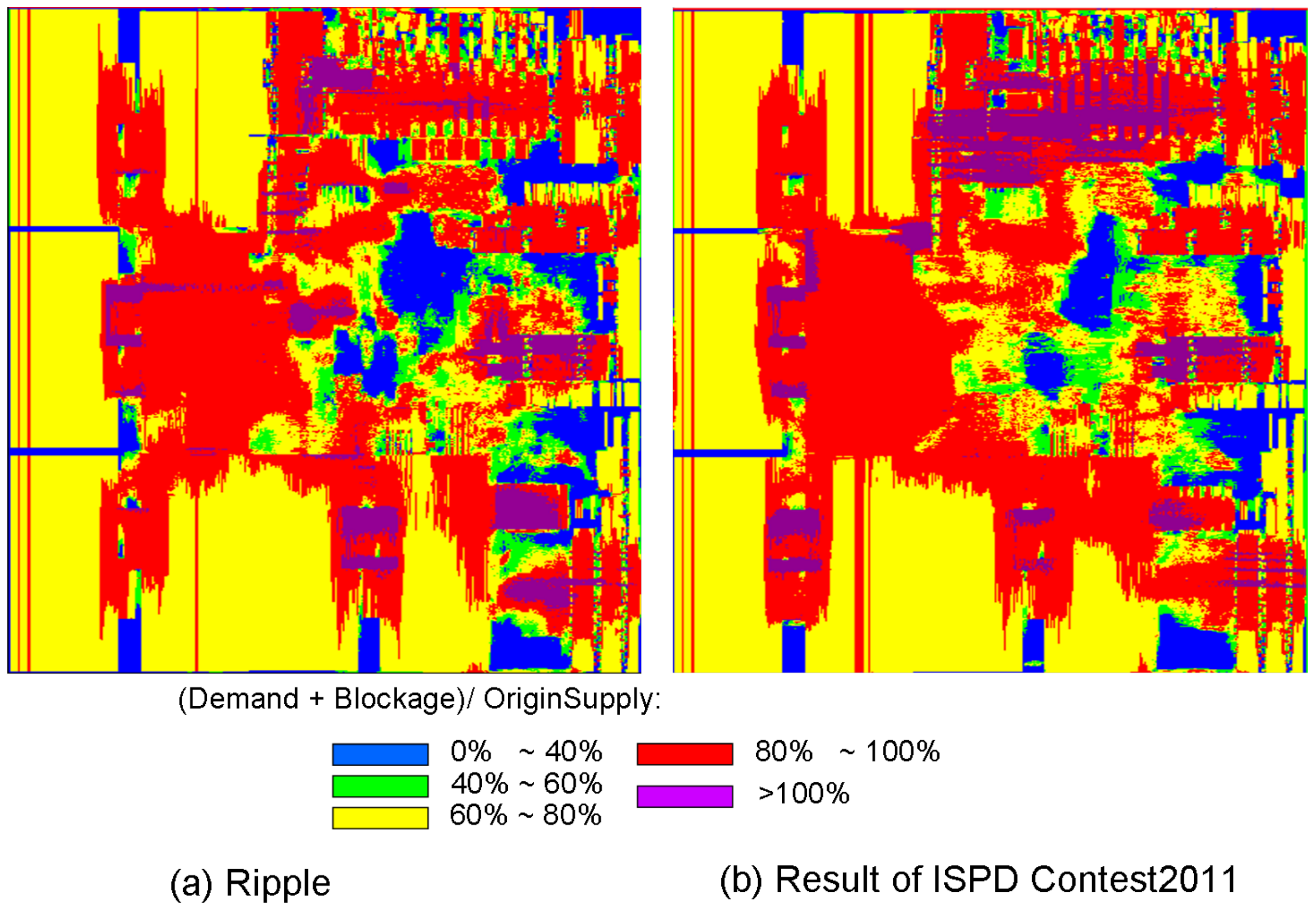


Figure 6.2: Lower-upper-bound HPWL, inflation area and overflow at each iteration (superblue18).



(a) Ripple

(b) Result of ISPD Contest2011

Figure 6.3: Congestion maps for RippleBasic solution and the top result in ISPD 2011 Contest on superblue2.

Table 6.7: Runtime for global and detailed placement

Benchmark	Global (s)	Detailed (s)	Total (s)
sb1	2798	209	3007
sb2	4764	312	5076
sb4	1819	225	2044
sb5	2888	278	3166
sb10	6056	442	6498
sb12	5990	371	6361
sb15	3847	550	4397
sb18	2168	201	2369
Avg.	92.14%	7.86%	-

method called NoScale to do comparison. NoScale is similar to RippleBasic, but it only removes tiles from the head of $InflationList$ one by one without reducing the inflation ratios of the remaining tiles in $InflationList$. Since the overflow obtained by NCTUgr [34] is independent of the machine used, we use NCTUgr to do global routing.

The overflow in Table 6.8 is the total overflow by considering the wire width and wire space on different metal layers. Comparing with RippleBasic, the total overflow of the top results in the ISPD 2011 Contest and SimPLR are larger by 47.48% and 28.30% respectively. Besides, the HPWL in the top results of the ISPD 2011 Contest and SimPLR are larger by 5.26% and 3.96% respectively. RoutedWL in Table 6.8 is the number of GCells passed through by the routed wires, and the result of RippleBasic is also shorter than the top results of the ISPD 2011 Contest and SimPLR. The runtime in Table 6.8 is obtained from the ISPD 2011 Contest [1] and SimPLR [29]. Since the runtime depends on the machine, we do not compare it in Table 6.8. In Table 6.8, we can also see that the total overflow, HPWL and RoutedWL are better in the results of RippleBasic than that in NoScaled.

Although both Ripple and SimPLR use the lower-upper bound framework in global placement, the main idea to handle congestion is different. A global router

Table 6.8: Comparisons with the top results of the ISPD 2011 Contest [1] and SimPLR [29]

Benchmark	Method	Overflow	ACE (%)				HPWL	RoutedWL	Runtime
			0.50	1.00	2.00	5.00			(seconds)
sb1	RippleBasic	7928	101.66	100.83	100.41	100.17	270288485	9524748	3007
	NoScaled	7928	101.66	100.83	100.41	100.17	270288485	9524748	2920
	SimPLR	8056	102.71	101.35	100.68	100.27	279004812	9835021	3101
	ISPD2011	12514	103.30	101.65	100.82	100.33	286630036	10124374	5686
sb2	RippleBasic	215112	111.78	109.06	104.88	101.95	631736150	21547075	5076
	NoScaled	224182	113.06	109.29	104.94	101.98	633737957	21684362	5295
	SimPLR	297914	119.82	113.15	107.13	102.85	660940059	22644961	6498
	ISPD2011	306748	111.85	109.25	105.13	102.05	704080370	24139506	16887
sb4	RippleBasic	25266	106.40	103.20	101.60	100.64	218347584	6929099	2044
	NoScaled	23044	105.90	102.95	101.48	100.59	218607769	6920948	2103
	SimPLR	9220	102.37	101.19	100.59	100.24	231444466	7209504	1487
	ISPD2011	33706	110.82	106.08	103.04	101.22	231798409	7319250	7631
sb5	RippleBasic	54472	104.51	102.26	101.13	100.45	343406094	11845978	3166
	NoScaled	54472	104.51	102.26	101.13	100.45	343406094	11845978	3222
	SimPLR	93222	110.91	105.45	102.73	101.09	355052381	12234359	3110
	ISPD2011	52488	106.40	103.20	101.60	100.64	359229472	12352907	10885
sb10	RippleBasic	49470	104.87	102.43	101.22	100.49	576116390	19544587	6498
	NoScaled	66922	106.24	103.12	101.56	100.62	576860165	19610052	6887
	SimPLR	81042	111.04	106.46	103.23	101.29	592176549	20021886	4400
	ISPD2011	75972	107.09	104.18	102.09	100.84	563980408	18542473	12873
sb12	RippleBasic	41282	108.59	106.79	103.86	101.54	364157547	13101209	6361
	NoScaled	55092	109.78	107.39	105.00	102.00	367977525	13317779	6958
	SimPLR	29172	108.44	105.27	102.63	101.05	377265823	13383941	2599
	ISPD2011	101702	110.46	110.23	108.94	104.71	375222962	13373492	10046
sb15	RippleBasic	43300	113.39	109.07	104.53	101.81	314006505	10001805	4397
	NoScaled	152412	120.24	115.12	111.61	104.89	317220427	10147739	5456
	SimPLR	47014	112.00	107.76	103.88	101.55	337955501	10812010	2600
	ISPD2011	56882	109.28	105.65	102.82	101.13	360309184	11471482	17831
sb18	RippleBasic	11060	104.20	102.10	101.05	100.42	167174630	6514009	2369
	NoScaled	15224	106.00	103.00	101.50	100.60	166951489	6622267	2466
	SimPLR	8988	102.74	101.37	100.69	100.27	165750027	6699189	1283
	ISPD2011	20532	109.11	104.61	102.31	100.92	155606626	5944270	2413
Avg.	RippleBasic	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	NoScaled	+33.80%	+1.40%	+0.98%	+1.09%	+0.47%	+0.34%	+0.67%	+7.26%
	SimPLR	+28.30%	+1.71%	+0.75%	+0.35%	+0.14%	+3.96%	+3.87%	N/A
	ISPD2011	+47.48%	+1.51%	+1.09%	+0.99%	+0.54%	+5.26%	+4.30%	N/A

BFG-R [22] is used in SimPLR to obtain accurate congestion map and the past cell inflated size will be remembered. Therefore, the number of times to call congestion estimation and cell inflation is not large. In Ripple, although our congestion map is obtained by some probabilistic method, we do a lot of estimations (more than 100 times) on intermediate layouts and use cell inflation on current layout without remembering the past inflated size. Moreover, different with SimPLR that adjusts target density during placement, we only use cell inflation in congested region without changing the target density of the whole chip. Although our runtime is longer because of the many times of congestion estimation done, we can obtain shorter HPWL in most benchmarks, while the overflow is also smaller.

Since the total overflow do not provide an accurate view of design routability [60], we use another metric called ACE (Average Congestion of GCell Edges) [60] to do evaluation. The congestion (in percentage) of a GCell edge e is $Cong(e) = 100 \times (w_e + b_e)/c_e$, where c_e is the total routing capacity of edge e, w_e is the routing demand or wire occupancy on edge e, and b_e is the routing blockage on edge e. The ACE metric is based on the distribution of the g-edge congestion. The $ACE(x)$ is the average congestion of the top $x\%$ congested g-edges. Here, the $ACE(x)$, where $x \in \{0.5, 1, 2, 5\}$, is used to provide a more accurate view of the design congestion.

From Table 6.8, we can see that in most benchmarks, we obtain the smallest ACE. The congestion of Ripple can always be spread more evenly. An example layout is shown in Fig. 6.3. The regions colored purple, with congestion > 100%[1], are relatively less in the RippleBasic congestion maps.

[1] Can be seen more clearly in color.

6.4.2 Overall performance of Ripple — an integration of different techniques

Based on the framework shown in Fig. 3.1, we use Simplified FastRoute to do lookahead routing analyze during global placement. Besides, the techniques of routing path-based cell inflation and congested cluster optimization are applied. The post-process SRP is also invoked after detailed placement. All the benchmarks were run on an Intel Xeon Linux workstation with 3.40GHz and 32 GB RAM. The target density is set to 0.95 for all the benchmarks. The latest benchmarks (in the ICCAD 2012 contest) are used in this section.

Routability is measured by scaled wirelength (ScaledWL) computed in the same way as in the ICCAD contest 2012 [55]. Both HPWL and congestion are considered in ScaledWL. After obtaining the routing result by NCTUgr [34], the ACE metric [60], which is based on the distribution of the GCell congestion, is used to evaluate congestion where $ACE(x), x \in \{0.5, 1, 2, 5\}$, is the average congestion of the top $x\%$ congested GCell edges.

Routing path-based cell inflation and spreading (RPB) and **congested cluster improvement (CCI)** are evaluated by comparing with the placer using probabilistic congestion estimation [18] without these techniques (noRPB/CCI). We report the (1) $ACE(x)$ where $x \in (0.5, 1, 2, 5)$, (2) HPWL, (3) ScaledWL and (4) Runtime in Table 6.9. We can see that most of the runtime is spend on the technique of CCI. Both RPB and CCI can improve congestion and scaled wirelength obviously.

Routability comparisons with the top results of the ICCAD 2012 Contest is shown in Table 6.10. In terms of scaled wirelength, Ripple outperforms the others on average. Since different machines were used and we did not use multi-threading.

The runtimes reported in Table 6.10 are just for reference and cannot be directly compare.

Table 6.9: Evaluation of routing path-based cell inflation and spreading (RPB) and congested cluster improvement (CCI)

Benchmark	Method	ACE (%)				HPWL	ScaledWL	Runtime
		0.50	1.00	2.00	5.00			(seconds)
sb1	noRPB/CCI	111.78	110.43	107.57	103.03	272906304	340071852	1989
	RPB	118.59	115.79	112.48	106.53	276418525	387109145	1636
	RPB+CCI	102.63	101.31	100.66	100.26	278613308	288763344	5462
sb3	noRPB/CCI	129.61	122.59	117.29	110.89	307528119	492931490	2646
	RPB	111.26	108.96	106.19	102.47	343317959	417703300	2584
	RPB+CCI	105.29	103.19	101.59	100.64	333315621	360085825	8487
sb4	noRPB/CCI	106.06	103.03	101.51	100.61	218230511	236570795	1423
	RPB	105.18	102.59	101.3	100.52	219891835	235700081	1127
	RPB+CCI	102.75	101.37	100.69	100.27	218321746	226639292	4504
sb5	noRPB/CCI	118.04	114.54	109.83	103.93	335332413	451886534	2178
	RPB	104.87	102.43	101.22	100.49	345713557	369066484	1543
	RPB+CCI	100.79	100.4	100.2	100.08	344935609	348733540	5728
sb7	noRPB/CCI	111.04	108.81	106.75	103.11	395288349	483366222	4274
	RPB	102.97	101.48	100.74	100.3	425010166	442509990	3928
	RPB+CCI	101.91	100.96	100.48	100.19	417956482	429041785	11387
sb10	noRPB/CCI	117.63	114.65	110.66	105.72	565020331	771229468	5868
	RPB	105.19	102.95	101.47	100.59	582988405	627566932	4393
	RPB+CCI	101.79	100.9	100.45	100.18	583014725	597515266	14383
sb16	noRPB/CCI	115.36	114.35	113.18	109.46	249202445	347035229	1470
	RPB	108.03	104.23	102.11	100.85	263035858	293059642	1267
	RPB+CCI	104.48	102.24	101.12	100.45	267097907	283689195	4942
sb18	noRPB/CCI	115.95	112.97	110.2	105.2	171609483	228658479	2061
	RPB	112.19	111.09	108.84	104.41	170358833	217035361	1945
	RPB+CCI	107.18	103.88	101.94	100.78	166702202	183918099	6267
Avg.	noRPB/CCI	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	RPB	-6.18%	-5.75%	-4.86%	-3.06%	+4.44%	-10.80%	-15.91%
	RPB+CCI	-10.66%	-9.67%	-7.97%	-4.64%	+3.77%	-18.90%	+179.15%

Table 6.10: Comparison with the top results of the ICCAD 2012 contest [55]

(The runtime (s) of VDAPlacer, SimPLR, NTUPlacer4h are obtained from the ICCAD 2012 contest.)

	VDAPlace		SimPLR		NTUPlace4h		Ripple	
Benchmark	ScaledWL	Runtime	ScaledWL	Runtime	ScaledWL	Runtime	ScaledWL	Runtime
sb1	369837790	885	278880496	2319	284990372	8769	288763344	5462
sb3	631616365	1173	343918131	2706	447690064	7193	360085825	8487
sb4	282282824	604	243416746	1267	236022004	4866	226639292	4504
sb5	569553239	874	360305914	2154	421690925	7322	348733540	5728
sb7	500120583	1725	431325680	3249	413677502	15005	429041785	11387
sb10	1036030901	1316	690852663	4837	718978960	12352	597515266	14383
sb16	322305959	691	285718339	1797	283297667	6024	283689195	4942
sb18	291093797	658	182345027	1645	170935389	4622	183918099	6267
Avg.	+47.25%	N/A	+3.62%	N/A	+9.52%	N/A	1.00	N/A

CHAPTER 7

Conclusion

Today, the dominance of interconnect in area, delay and power is increasing. Traditional placers focus on minimizing wirelength. However, wirelength-driven placers may generate hard-to-route solutions and create a lot of troubles for the downstream routing process. The mismatch between the objective of wirelength and routing congestion makes the routability issue become more and more important in placement.

In this thesis, each stage in the placement process, including global placement, legalization, detailed placement and post-process, are studied carefully to trade off between routability and wirelength. During global placement, several techniques are studied and developed: (1) routing congestion analysis by probabilistic method and Simplified FastRoute with pin density consideration, (2) dynamic cell inflation adjustment, (3) routing path-based cell inflation and spreading, and (4) robust congested cluster optimization. After obtaining a competitive global placement result, we propose many simple and effective way to avoid worsening congestion while optimizing wirelength during legalization and detailed placement, such as (1) limiting the maximum length of a segment in displacement-driven legalization, (2) determining the stopping criteria in each step of detailed placement. All these techniques are studied and evaluated by experimental analysis. For post-processing, we propose a refinement tool called SRP that is independent of any placer and global router. Based on a given placement layout and global routing result, SRP relocates problematic cells by considering routing and placement simultaneously. Not only overflow from local nets, but overflow from global and semi-global nets can be mitigated by SRP. A cell will be relocated while its associated nets will be rerouted if its connections go across any congested region, even if the cell itself is not in a congested region. A combination of these researches provides a powerful tool Ripple for solving the routability placement problem. We believe that our study will be useful for this routability-driven placement problem in both industry and academia.

As process technology advances, the numbers of circuit elements and interconnections in a design can easily reach tens of millions and the numbers are still growing. Highly efficient routability optimization method during placement is still in great demand. Besides, future research on placement should also address other requirements in VLSI design, such as power reduction, timing constraints, and the manufacturability issues.

Bibliography

[1] *ISPD 2011 Routability-Driven Placement Contest. http://www.ispd.cc/contests/11/ispd2011_contest.html*. xvii, 114, 117, 118

[2] C.J. Alpert, Z. Li, M.D. Moffitt, G.J. Nam, J.A. Roy, and G. Tellez. What makes a design difficult to route. In *ISPD*, pages 7–12. ACM, 2010. 8, 33

[3] C.J. Alpert, D.P. Mehta, and S.S. Sapatnekar. *Handbook of algorithms for physical design automation*. CRC Press Company, Incorporated, 2009. xi, 7, 13, 24, 25

[4] U. Brenner and A. Rohe. An effective congestion-driven placement framework. *TCAD*, 22(4):387–394, 2003. 26, 31, 33, 34, 65

[5] Andrew E Caldwell, Andrew B Kahng, Stefanus Mantik, Igor L Markov, and Alexander Zelikovsky. On wirelength estimations for row-based placement. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 18(9):1265–1278, 1999. 26

[6] T.F. Chan, J. Cong, J.R. Shinnerl, K. Sze, and M. Xie. mpl6: Enhanced multilevel mixed-size placement. In *ISPD*, pages 212–214. ACM, 2006. 46, 111

[7] Tony Chan, Jason Cong, and Kenton Sze. Multilevel generalized force-directed method for circuit placement. In *Proceedings of the 2005 international symposium on Physical design*, pages 185–192. ACM, 2005. 24

[8] Tung-Chieh Chen, Zhe-Wei Jiang, Tien-Chang Hsu, Hsin-Chen Chen, and Yao-Wen Chang. A high-quality mixed-size analytical placer considering

preplaced blocks and density constraints. In *Computer-Aided Design, 2006. ICCAD'06. IEEE/ACM International Conference on*, pages 187–192. IEEE, 2006. 24, 43

[9] C. Chu. FLUTE: fast lookup table based wirelength estimation technique. In *ICCAD*, pages 696–701. IEEE Computer Society, 2004. 61

[10] Chris Chu and Yiu-Chung Wong. Fast and accurate rectilinear steiner minimal tree algorithm for vlsi design. In *Proceedings of the 2005 international symposium on Physical design*, pages 28–35. ACM, 2005. 29, 45, 46

[11] Y.L. Chuang and et al. Design-hierarchy aware mixed-size placement for routability optimization. In *ICCAD*, pages 663–668. IEEE, 2010. 31, 35, 36, 37

[12] Jason Cong, Guojie Luo, Kalliopi Tsota, and Bingjun Xiao. Optimizing routability in large-scale mixed-size placement. In *ASP-DAC*, 2013. 31, 38, 46, 47, 50

[13] Jeffrey A Davis, Vivek K De, and James D Meindl. A stochastic wire-length distribution for gigascale integration (gsi). i. derivation and validation. *Electron Devices, IEEE Transactions on*, 45(3):580–589, 1998. 27

[14] W Donath. Placement and average interconnection lengths of computer logic. *Circuits and Systems, IEEE Transactions on*, 26(4):272–277, 1979. 27

[15] Rafael C Gonzales and RE Woods. Digital image processing, 1993. 39, 45

[16] Satoshi Goto. An efficient algorithm for the two-dimensional placement problem in electrical circuit layout. *Circuits and Systems, IEEE Transactions on*, 28(1):12–18, 1981. 79

[17] X. He, T. Huang, Chow W., Kuang J., Lam K., Cai W., and E.F.Y. Young. Ripple 2.0: High quality routability-driven placement via global router integration. In *DAC*, 2013. 26, 31

[18] X. He, T. Huang, L. Xiao, H. Tian, G. Cui, and E.F.Y. Young. Ripple: An effective routability-driven placer by iterative cell movement. In *ICCAD*, pages 74–79, 2011. xi, 11, 12, 31, 46, 47, 72, 77, 88, 111, 120

[19] W. Hou and et al. A new congestion-driven placement algorithm based on cell inflation. In *ASP-DAC*, pages 605–608. IEEE, 2001. 65

[20] Meng-Kai Hsu, Yi-Fang Chen, Chau-Chin Huang, Tung-Chieh Chen, and Yao-Wen Chang. Routability-driven placement for hierarchical mixed-size circuit designs. In *Proceedings of the 50th Annual Design Automation Conference*, page 151. ACM, 2013. xi, xii, 31, 36, 37, 38, 39, 44

[21] M.K. Hsu, S. Chou, T.H. Lin, and Y.W. Chang. Routability-driven analytical placement for mixed-size circuit designs. In *ICCAD*, pages 80–84. IEEE, 2011. 31, 34, 35, 46, 77, 88, 111

[22] J. Hu and et al. Completing high-quality global routes. In *ISPD*, pages 35–41, 2010. 26, 41, 55, 119

[23] Jin Hu, Myung-Chul Kim, and Igor L Markov. Taming the complexity of coordinated place and route. In *Proceedings of the 50th Annual Design Automation Conference*, page 150. ACM, 2013. 31, 39, 41, 42

[24] Devang Jariwala and John Lillis. Rbi: Simultaneous placement and routing optimization technique. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(1):127–141, 2007. 31

[25] Z.W. Jiang and et al. Routability-driven analytical placement by net overlapping removal for large-scale mixed-size designs. In *DAC*, pages 167–172, 2008. 31, 35

[26] T. Jindal and et al. Detecting tangled logic structures in vlsi netlists. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 603–608. IEEE, 2010. xii, 47, 48, 73

[27] A.B. Kahng and Q. Wang. Implementation and extensibility of an analytic placer. *TCAD*, 24(5):734–747, 2005. 24, 35, 43

[28] A.B. Kahng and X. Xu. Accurate pseudo-constructive wirelength and congestion estimation. In *SLIP*, pages 61–68. ACM, 2003. 26, 55

[29] M.C. Kim, J. Hu, D.J. Lee, and I.L. Markov. A simplr method for routability-driven placement. In *ICCAD*, pages 67–73. IEEE Press, 2011. xi, xii, xvii, 11, 12, 26, 31, 39, 41, 42, 77, 88, 111, 114, 117, 118

[30] M.C. Kim, D.J. Lee, and I.L. Markov. simpl: an effective placement algorithm. *TCAD*, 31(1):50–60, 2012. xii, 14, 24, 39, 40, 53, 68

[31] Bernard S Landman and Roy L Russo. On a pin versus block relationship for partitions of logic graphs. *Computers, IEEE Transactions on*, 100(12):1469–1479, 1971. 27

[32] C. Li, M. Xie, C.K. Koh, J. Cong, and P.H. Madden. Routability-driven placement and white space allocation. *TCAD*, 26(5):858–871, 2007. 31, 32, 98

[33] Zhuoyuan Li, Weimin Wu, and Xianlong Hong. Congestion driven incremental placement algorithm for standard cell layout. In *Proceedings of the*

2003 Asia and South Pacific Design Automation Conference, pages 723–728. ACM, 2003. 31

[34] W.H. Liu, W.C. Kao, Y.L. Li, and K.Y. Chao. Multi-threaded collision-aware global routing with bounded-length maze routing. In *DAC*, pages 200–205. ACM, 2010. xii, xiii, 26, 38, 73, 74, 93, 111, 117, 120

[35] Jinan Lou, Shashidhar Thakur, Shankar Krishnamoorthy, and Henry S Sheng. Estimating routing congestion using probabilistic analysis. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 21(1):32–41, 2002. 26, 28

[36] Gi-Joon Nam. Ispd 2006 placement contest: Benchmark suite and results. In *Proceedings of the 2006 international symposium on Physical design*, pages 167–167. ACM, 2006. 10

[37] Gi-Joon Nam, Charles J Alpert, Paul Villarrubia, Bruce Winter, and Mehmet Yildiz. The ispd2005 placement contest and benchmark suite. In *Proceedings of the 2005 international symposium on Physical design*, pages 216–220. ACM, 2005. 10

[38] G.J. Nam and J. Cong. *Modern circuit placement: best practices and results*. Springer Publishing Company, Incorporated, 2007. xi, 7, 10, 21, 22, 46, 71

[39] M. Pan and C. Chu. Fastroute: A step to integrate global routing into placement. In *ICCAD*, pages 464–471. ACM, 2006. 29, 55, 63, 64, 85

[40] M. Pan, N. Viswanathan, and C. Chu. An efficient and effective detailed placement algorithm. In *ICCAD*, pages 48–55. IEEE Computer Society, 2005. 77, 78, 79, 81, 82, 83, 84

[41] Min Pan and Chris Chu. Fastroute 2.0: A high-quality and efficient global router. In *Proceedings of the 2007 Asia and South Pacific Design Automation Conference*, pages 250–255. IEEE Computer Society, 2007. 29, 30

[42] Min Pan and Chris Chu. Ipr: An integrated placement and routing algorithm. In *Proceedings of the 44th annual Design Automation Conference*, pages 59–62. ACM, 2007. 26, 31, 77, 98

[43] P.N. Parakh, R.B. Brown, and K.A. Sakallah. Congestion driven quadratic placement. In *DAC*, pages 275–278. ACM, 1998. 31

[44] J.A. Roy and et al. CRISP: congestion reduction by iterated spreading during placement. In *ICCAD*, pages 357–362. ACM, 2009. xi, 31, 33, 34, 64, 65, 91, 92

[45] J.A. Roy and I.L. Markov. Seeing the forest and the trees: Steiner wirelength optimization in placement. *TCAD*, 26(4):632–644, 2007. 26, 31, 98

[46] C. Sham and E.F.Y. Young. Congestion prediction in early stages. In *SLIP*, pages 91–98. ACM, 2005. 26, 28, 55, 61

[47] N. A. Sherwani. *Algorithms for physical design automation / 3rd edition*. Springer-Verlag New York, LLC, 1999. xi, 3, 4

[48] H. Shojaei, A. Davoodi, and J.T. Linderoth. Congestion analysis for global routing via integer programming. In *ICCAD*, pages 256–262. IEEE, 2010. 10, 57

[49] P. Spindler and et al. Kraftwerk2: A fast force-directed quadratic placement approach using an accurate net model. *TCAD, 27(8)*, pages 1398–1411, 2008. 23, 39, 52

[50] P. Spindler and F.M. Johannes. Fast and accurate routing demand estimation for efficient routability-driven placement. In *DATE*, pages 1–6. IEEE, 2007. 26, 27, 31, 35, 55, 56

[51] Dirk Stroobandt and Jan Van Campenhout. Accurate interconnection length estimations for predictions early in the design cycle. *VLSI Design*, 10(1):1–20, 1999. 27

[52] N. Viswanathan, C. Alpert, C. Sze, Z. Li, and Y. Wei. The dac 2012 routability-driven placement contest and benchmark suite. In *DAC*, pages 774–782. ACM, 2012. xi, xii, 8, 11, 12, 17, 38, 107, 111

[53] N. Viswanathan and C.C.N. Chu. Fastplace: Efficient analytical placement using cell shifting, iterative local refinement, and a hybrid net model. *TCAD*, 24(5):722–733, 2005. 24

[54] N. Viswanathan and et al. The ispd-2011 routability-driven placement contest and benchmark suite. In *ISPD*, pages 141–146. ACM, 2011. xv, 8, 17, 107, 108

[55] Natarajan Viswanathan and et al. Iccad-2012 cad contest in design hierarchy aware routability-driven placement and benchmark suite. In *ICCAD*, pages 345–348, 2012. xvii, 8, 12, 17, 36, 107, 120, 123

[56] Natarajan Viswanathan, Min Pan, and Chris Chu. Fastplace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control. In *Proceedings of the 2007 Asia and South Pacific Design Automation Conference*, pages 135–140. IEEE Computer Society, 2007. 31

[57] Maogang Wang and Majid Sarrafzadeh. Modeling and minimization of routing congestion. In *Proceedings of the 2000 Asia and South Pacific Design Automation Conference*, pages 185–190. ACM, 2000. 31

[58] Maogang Wang, Xiaojian Yang, Kenneth Eguro, and Majid Sarrafzadeh. Multi-center congestion estimation and minimization during placement. In *Proceedings of the 2000 international symposium on Physical design*, pages 147–152. ACM, 2000. 26

[59] Maogang Wang, Xiaojian Yang, and Majid Sarrafzadeh. Congestion minimization during placement. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 19(10):1140–1148, 2000. 31

[60] Y. Wei and et al. Glare: Global and local wiring aware routability evaluation. In *DAC*, pages 768–773. ACM, 2012. 10, 119, 120

[61] J. Westra, C. Bartels, and P. Groeneveld. Probabilistic congestion prediction. In *ISPD*, pages 204–209. ACM, 2004. xi, 26, 28, 29, 55, 61

[62] Jurjen Westra and Patrick Groeneveld. Is probabilistic congestion estimation worthwhile? In *Proceedings of the 2005 international workshop on System level interconnect prediction*, pages 99–106. ACM, 2005. 26

[63] Yue Xu, Yanheng Zhang, and Chris Chu. Fastroute 4.0: global router with efficient via minimization. In *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, pages 576–581. IEEE Press, 2009. 26

[64] X. Yang, B.K. Choi, and M. Sarrafzadeh. Routability-driven white space allocation for fixed-die standard-cell placement. *TCAD*, 22(4):410–419, 2003. 31, 32, 98

[65] Xiaojian Yang, Ryan Kastner, and Majid Sarrafzadeh. Congestion estimation during top-down placement. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 21(1):72–80, 2002. 26

[66] Y. Zhang and C. Chu. Crop: Fast and effective congestion refinement of placement. In *ICCAD*, pages 344–350. IEEE, 2009. 31, 77, 91, 92