2011

Handling the complexity of routing problem in modern VLSI design

Yanheng Zhang
Iowa State University

Handling the complexity of routing problem in modern VLSI design

by

Yanheng Zhang

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Program of Study Committee:
Chris Chu, Major Professor
Randall L. Geiger
Akhilesh Tyagi
Degang Chen
Yong Guan

Iowa State University

Ames, Iowa

2010

DEDICATION

To my parents and my wife

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

In VLSI physical design, the routing task consists of using over-the-cell metal wires to connect pins and ports of circuit gates and blocks. Traditionally, VLSI routing is an important design step in the sense that the quality of routing solution has great impact on various design metrics such circuit timing, power consumption, chip reliability and manufacturability etc. As the advancing VLSI design enters the nanometer era, the routing success (routability issue) has been arising as one of the most critical problems in back-end design. In one aspect, the degree of design complexity is increasing dramatically as more and more modules are integrated onto the chip. Much higher chip density leads to higher routing demands and potentially more risks in routing failure. In another aspect, with decreasing design feature size, there are more complex design rules imposed to ensure manufacturability. These design rules are hard to satisfy and they usually create more barriers for achieving routing closure (i.e., generate DRC free routing solution) and thus affect chip time to market (TTM) plan.

In general, the behavior and performance of routing are affected by three consecutive phases: placement phase, global routing phase and detailed routing phase in a typical VLSI physical design flow. Traditional CAD tools handle each of the three phases independently and the global picture of the routability issue is neglected. Different from conventional approaches which propose tools and algorithms for one particular design phase, this thesis investigates the routability issue from all three phases and proposes a series of systematic solutions to build a more generic flow and improve quality of results (QoR).

For the placement phase, we will introduce a mixed-sized placement refinement tool for alleviating congestion after placement. The tool shifts and relocates modules based on a global routing estimation. For the global routing phase, a very fast and effective global router is

developed. Its performance surpasses many peer works as verified by ISPD 2008 global routing contest results. In the detailed routing phase, a tool is proposed to perform detailed routing using regular routing patterns based on a correct-by-construction methodology to improve routability as well as satisfy most design rules. Finally, the tool which integrates global routing and detailed routing is developed to remedy the inconsistency between global routing and detailed routing.

To verify the algorithms we proposed, three sets of testcases derived from ISPD98 and ISPD05/06 placement benchmark suites are proposed. The results indicate that our proposed methods construct an integrated and systematic flow for routability improvement which is better than conventional methods.

CHAPTER 1. GENERAL INTRODUCTION

Introduction

The physical design of very large scale integrated circuits (VLSI) primarily consists the tasks of realizing the size and location of all contained logic modules (placement) and the wires connecting pins and ports (routing). Traditionally, VLSI routing has been an important problem in VLSI back-end design since the quality of routing results has great impact on various design metrics such as circuit timing, power consumption, chip reliability and manufacturability etc. With the advancing fabrication technology enters the nanometer scale, the physical design, especially VLSI routing (routability issue) has been facing more challenges. The challenges come from two parts. First, with much higher density on single chip, the problem size of routing is growing in an exponential rate. With more and more modules integrated into the chip, the routing requirement for connecting these modules is also growing dramatically. The routing problem is becoming harder to handle. Second, with diminishing feature size, more design constraints and design rules need be respected. For instance, there are many complex design rules that are imposed to ensure manufacturability. It has been reported that for 32nm process, the number of design rules reaches several thousand[1]. And the design rule manual has reached approximately one thousand pages [2]. Because of these challenges, the design closure can be seriously affected because of the delayed routing closure. The chip time to market (TTM) cycle can also be significantly prolonged due to unaccomplished routing mission.

VLSI routing problem is NP-hard[3], even for the simplest case with only a few nets and pins. To handle this problem, people usually divide the routing process to two phases: global routing and detailed routing. In global routing phase, chip layout is partitioned into a set of regular global routing cells (G-Cells) and global routing grid graph is constructed. Rough

routing decision will be made on a G-Cell to G-Cell connection on the global routing grid graph. The subsequent detailed routing is formulated to route all nets defined in netlist based on the provided global routing solution. It realizes the exact routing paths considering geometrical constraints as well as various design rules.

Traditionally, routing problem is handled by providing good solution either to global routing phase or detailed routing phase. There have been many researches conducted for global routing or detailed routing respectively. In particular, for global routing, the most popular approach is iterative ripup and reroute based. The approach first breaks each multi-pin net into a set of two-pin nets. Then the two-pin nets are routed in a sequential order. The routing solution is iteratively refined by ripup and reroute until it reaches an acceptable solution quality in terms of overflow. There have been many works proposed in the past for boosting the ripup and reroute approach including works such as [4, 5, 6, 7, 8]. The two consecutive ISPD global routing contests [28, 29] have proved the dominance of sequential router and many new routers such as Archer[9], BoxRouter2.0 [10], FGR [11] and NTHU-R[12] are presented. Most of these routers adopt the history based idea[25] when performing ripup and reroute. In terms of detailed routing, it has been a traditional EDA problem since 70's when early works including the left edge algorithm[13] and the algorithm to handle channel routing [14] are presented. There are many new approaches that have been proposed during last two decades. For instance, DUNE[15] and MR[16] proposed multi-level process to handle gridless routing. The work in [17] proposed routing FPGA using boolean satisfiability. In track routing[18], an intermediate step between global routing and detailed routing is proposed. And a latest work [19] presented a meaningful technique to tackle the escape routing for dense pin clusters, which is the bottleneck of detailed routing.

However, considering the size and complexity of modern design, it becomes insufficient to handle the routability issue from the two routing phases independently. To facilitate developing better router and routing flow, it is necessary for better leveraging all routability related phases in physical design flow and remedying the inconsistencies between each phase. As shown in Figure 1.1, VLSI physical design typically contains partitioning, floorplanning, placement and

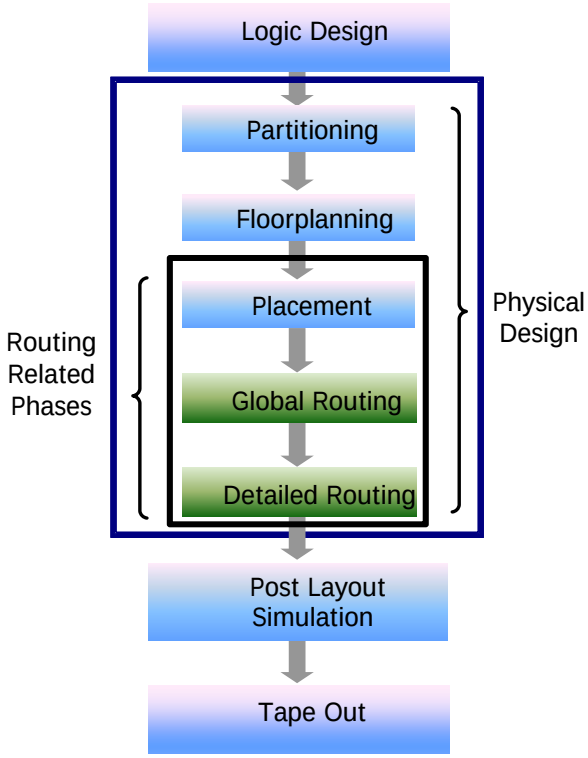

Figure 1.1 Typical VLSI physical design flow from logical design to tape out.

routing(global routing and detailed routing), which is after logical synthesis and before post layout simulation. Partitioning and floorplanning are early physical design phases which do not have great impact on routing performance because the size and position of each module is not yet determined. Placement, global routing and detailed routing are the phases that have close relationship with routability issue. Therefore, unlike traditional methods, this thesis will provide a series of systematic solutions for the three design phases to build a more generic and high-performance routing flow. With the tools we proposed, a lot of complicated design can be handled with ease and short runtime.

To assist the analysis for all the proposed tools and algorithms, we derive three sets of design testcases from ISPD98[20] and ISPD05/06 [21, 22] for performance verification. Experimental results reveal that the routing performance for each of the three phases can be greatly improved.

Thesis Organization

As mentioned in earlier section, this thesis will provide solutions for each of the three phases related to the routability problem. The organization of following chapters can be viewed in Figure 1.2.

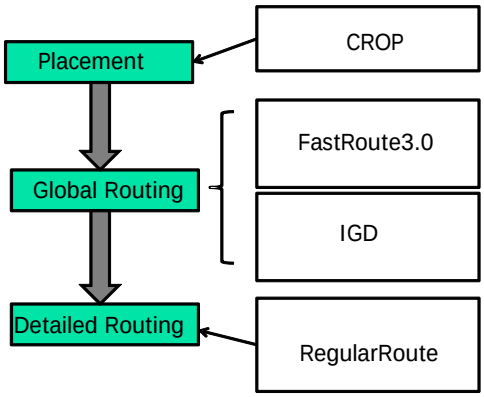

Figure 1.2 Thesis organization of following chapters.

In Chapter 2, we will introduce FastRoute 3.0, a very fast and efficient global router based on the virtual capacity idea. FastRoute 3.0 is the solution we provide for the global routing

stage. It is mainly based on the virtual capacity idea. The virtual capacity idea encourages the usage of less congested routing edges and avoids using the routing edges that are consistently congested. Besides the virtual capacity idea, we also include discussion for some solution quality improvement techniques such as via aware maze routing and adaptive maze function for reducing the number of vias and improving wirelength respectively. The global router can save considerable runtime spent on ripup and reroute over conventional maze routing techniques. It is also faster than most of its peer works as verified by ISPD 2008 global routing contests[23].

Next in Chapter 3, we will propose a systematic solution for placement phase. A tool called CROP is proposed to refine any legalized placement solution to improve routability. The tool is independent of any placer, which enables CROP to be a postprocessing technique for any placement tool. CROP interleves a congestion-driven module shifting and a congestion-driven detailed placement technique. The shifting technique targests at better allocating routing resources. The shifting in each direction (X's or Y's) is formulated as a linear program (LP) for resizing each G-Cell. It's then further simplified to solve a less expensive longest-path computation in a very fast runtime. The congestion-driven detailed placement is applied to better distribute the routing demands. It is realized by weighting the half-perimeter wirelength (HPWL) with congestion factor during detailed placement. The tool could handle placement testcases with movable and/or fixed macro blocks. And both the global routing phase routability and detailed routing routability are checked using an academic global router and an industrial routing tool respectively. The experiments show great routability improvement can be achieved by using CROP.

In Chapter 4, we move to the detailed routing phase and provide a systematic solution called RegularRoute for better routing success as well as design rule satisfaction. It is based on a correct-by-construction strategy. With the input 2-D global routing solution and underlying routing tracks. The detailed routing problem is solved in a layer by layer manner. For each layer, the routing tracks are partitioned and organized in panels. Assigning global segments in each panel is formulated as a Maximum Weighted Independent Set (MWIS) problem and solved panel by panel. The experiments demonstrate that RegularRoute surpasses the industrial

router which is frequently used by IC designers.

In Chapter 5, we will introduce a tool called IGD. It is the first algorithm ever which integrates global routing and detailed routing. The global router we proposed in Chapter 2 and the detailed router we proposed in Chapter 4 are integrated. The proposed method works on remedying the inconsistency between the two phases on setting the global edge capacity more precisely. In particular, the algorithm performs an initial capacity prediction to provide more accurate global edge capacity considering local pins, local connections and probabilistic pin promotion for each G-Cell. We also propose an iterative improvement framework to consistently improve the detailed routing solution in terms of the number of unassigned segments. The experimental results demonstrate that the algorithm can greatly reduce the number of unassigned segments.

In Chapter 6, general conclusion of the entire thesis will be made. And the future research recommendation will also be presented.

CHAPTER 2. FastRoute3.0: A Fast and High Quality Global Router Based on Virtual Capacity

A paper published in *International Conference on Computer-Aided Design*

Yanheng Zhang, Yue Xu and Chris Chu

Abstract

As an easily implemented approach, ripup and reroute has been employed by most of today's global routers, which iteratively applies maze routing to refine solution quality. But traditional maze routing is susceptible to get stuck at local optimal results. In this work, we will present a fast and high quality global router FastRoute3.0, with the new technique named virtual capacity. Virtual capacity is proposed to guide the global router at maze routing stage to achieve higher quality results in terms of overflow and runtime. During maze routing stage, virtual capacity works as a substitute for the real edge capacity in calculating the maze routing cost. There are two sub techniques included: (1) virtual capacity initialization, (2) virtual capacity update. Before the maze routing stage, FastRoute3.0 initializes the virtual capacity by subtracting the predicted overflow generated by adaptive congestion estimation (ACE) from the real edge capacity. And in the following maze routing iterations, we further reduce the virtual capacity by the amount of existing overflow(edge usage minus real edge capacity) for the edges that are still congested. To avoid excessive "pushing-away" of routing wires, the virtual capacity is increased by a fixed percentage of the existing overflow if edge usage is smaller than real edge capacity.

Experimental results show that FastRoute3.0 is highly proficient dealing with ISPD98, ISPD07 and ISPD08 benchmark suites. The results outperform published ripup and reroute

based academic global routers in both routability and runtime. In particular, (1) FastRoute3.0 completes routing all the ISPD98 benchmarks. (2) For ISPD07 and ISPD08 global routing contest benchmarks, it generates 12 out of 16 congestion free solutions. (3) The total runtime is enhanced greatly.

Introduction

As the feature size of modern VLSI design continues to shrink, the ascending circuit density poses greater challenges for modern chip routers. Modern designs are liable to congestion problems because of the increasingly concentrated routing demands. Besides, due to the rapidly growing problem size, the runtime required for completing a routing task is much longer than before.

placement stage and routing stage. Placement determines the instance and pin locations and hence the degree of difficulty for the routing problem that follows. Min and Chu [7] pointed out that to guide the placer to produce a routable placement, the routing estimation during placement should be consistent with the actual routing in the routing stage. It is desirable to have a fast router that can be repeatedly called right after the placement step for a quick estimation. The estimating router will be identical with the one used in the routing stage. From this sense, modern global routers need better take both runtime and routability issues into consideration.

In VLSI routing, global routing is an essential phase of the whole routing scheme, which determines routing usage based on an abstracted grid graph. Many global routing techniques have been proposed since the 1960s. The most popular global routing approach is iterative ripup and reroute based. The approach first breaks each multi-pin net into a set of two-pin nets. Then the two-pin nets are routed sequentially based on a predetermined order. The routing solution is iteratively refined by rip-up and reroute until reaching acceptable quality. However, the heuristic nature of such an approach is prone to getting stuck at local optimal solutions.

There have been several methods proposed to boost the quality of iterative ripup and reroute

approach. Kastner et al. [4] proposed a pattern based routing scheme. Hadsell and Madden [5] proposed to guide the routing by amplifying the congestion map with a new congestion cost function. BoxRouter [6] proposed a hybrid approach with the application of ILP to simultaneously handle multiple nets and achieved better routability. FastRoute [7] achieved very fast runtime by exploring congestion driven RSMT to avoid the extensive usage of maze routing. FastRoute2.0 [24] improved the solution quality over FastRoute by introducing monotonic routing and multi source multi sink maze routing. Recently, Archer [9], BoxRouter2.0 [10], FGR [11], and NTHU-R [12] are presented. All these four techniques employ the negotiation-based maze routing that was introduced by PathFinder [25]. Negotiation-based maze routing increases the maze routing cost for the edges that are consistently congested.

In this paper, we propose the virtual capacity technique which is a systematic way of tackling the congestion problem. As implied by the name, virtual capacity idea tries to guide ripup and reroute global router in the maze routing stage by the "virtual" capacities, instead of the real ones. Given a global routing solution, consider any congested routing edge e. Assume edge e has capacity c_e, routing demand u_e and overflow $o_e (= u_e - c_e > 0)$. The basic idea of virtual capacity is to reduce the capacity of e by o_e units (i.e., set the virtual capacity to $c_e - o_e$) and then run another round of global routing. Because of the reduction in capacity, edge e becomes more expensive to use and hence some of its routing demand will be pushed away. In the ideal situation, exactly o_e units of routing demand will be pushed away in order to bring the congestion back to the level of the previous round. Thus, the new routing demand will be $u_e - o_e = c_e$, i.e., the same as the original capacity. In order words, edge e will not be congested in the second round of global routing. In reality, less than o_e units of routing demand will get pushed away because other edges may not be willing to absorb the pushed routing demand. Nevertheless, virtual capacity is a systematic way to effectively reduce the overflow.

In FastRoute3.0, the virtual capacity is initialized by reducing the real capacity with the amount of estimated overflow generated by adaptive congestion estimation(ACE). During the following maze routing process, it is further reduced by the amount of existing overflow if the edge is still congested. Otherwise it is increased by timing a factor to prevent excessive

subtraction.

We develop FastRoute3.0 by integrating the new techniques into FastRoute2.0 [24], a fast rip-up and reroute based global router.

Compared with the other published academic global routers, FastRoute 3.0 achieves much better results in both routability and runtime. In particular, it completes routing all the ISPD98 benchmarks. For the ISPD07 and ISPD08 global routing contest benchmarks, it successfully generates 12 out of 16 congestion free solutions in very short runtime.

The next few sections are organized as follows: Section 2 describes the framework of FastRoute3.0. Section 3 presents the virtual capacity idea. Section 4 introduces two techniques that are effective in via reduction and convergence speedup. Section 5 discusses experimental results and comparisons. Conclusion is made in the section 6.

Materials and Methods

Preliminaries

Grid Graph Model

As is illustrated in figure 1, the whole routing region is partitioned into a number of global bins. Each global bin is represented by one node and each common boundary is represented by one edge in the grid graph. The edge is called global edge with the capacity of c_e. The overflow is defined as how much is usage u_e over the c_e. If u_e is smaller than c_e, $o_e = 0$, otherwise $o_e = u_e - c_e$.

Overview of FastRoute3.0

The flow of FastRoute3.0 is illustrated in figure 2. There are six major steps in the flow. Step 1, 3, 4 are techniques that we borrow from FastRoute2.0. The step 1 is multi-pin nets decomposition. We utilize FLUTE2.5 [26] to generate the congestion driven RSMT. Then the RSMT of all the multi-pin nets are decomposed into a group of two-pin nets. The step 2 is the

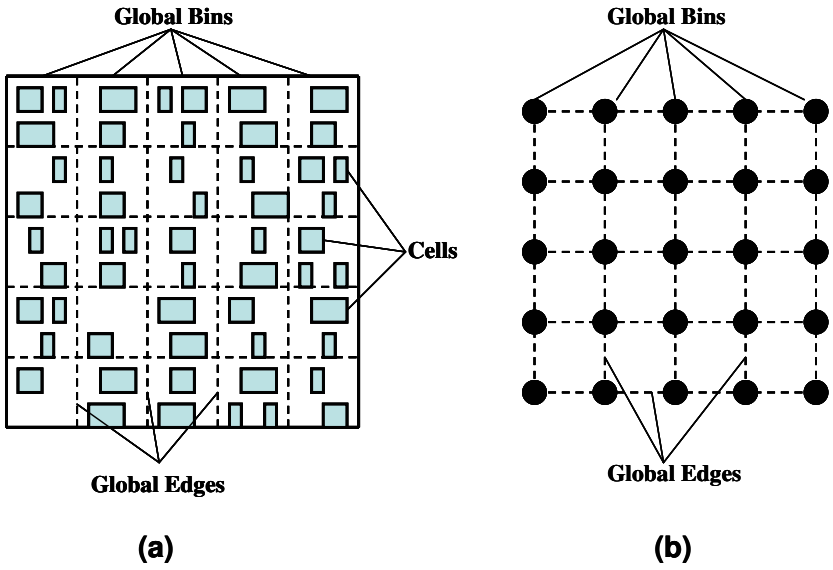

Figure 2.1 Global bins and corresponding global routing grid graph.

virtual capacity initialization technique which will be discussed in section 3. Step 3 is pattern routing. Normally L routing and Z routing are applied in this step. Step 4 is multi source multi sink maze routing. The maze routing cost is calculated by adaptive maze function based on virtual capacity. And step 5 is the virtual capacity update which is performed at the end of each maze routing iteration. The virtual capacity value will be either increased or decreased depends on current edge usage compared with original edge capacity. Step 4 and step 5 will be iteratively applied until the total overflow gets stuck. Step 6 performs layer assignment after the program runs out of the iterative ripup and reroute loop.

Virtual Capacity Technique

In this section, we will present the virtual capacity technique. Congestion reduction is key metric to evaluate a global router. Recent published academic global routers including [10], [11], [27] and [12] employ negotiation based maze routing technique, which increments the maze routing cost for edges that are consistently congested. In FastRoute3.0, we propose virtual capacity, an alternative but systematic method to handle congestion problem. Virtual capacity technique consists of two ideas. Section III.A discusses the virtual capacity initialization.

```
FastRoute3.0 Framework
begin
  Step1 : Congestion Driven Multi-Nets Decomposition
  Step2 : Virtual Capacity Initialization
  Step3 : Pattern Routing (L and Z Routing)
  while( total overflow not gets stuck)
    Step4 : Multi Sink Multi Source Maze Routing
    Step5 : Virtual Capacity Update
  end while
  Step6 : Layer Assignment
end
```

Figure 2.2 FastRoute3.0 framework

Section III.B describes the criteria how virtual capacity is updated.

Virtual Capacity Initialization by ACE

Virtual Capacity is initialized by subtracting the estimated overflow from the real edge capacity. In FastRoute3.0 we use adaptive congestion estimation(ACE) technique to predict the overflow. The idea is to assign net usage to proper routing edge and calculate the estimated overflow accordingly. Since before the maze routing, each decomposed two-pin net is routed without detour inside the bounding box. And the task of a global router, in general, is to distribute routing demand. The estimation is therefore based on the following two assumptions: (1) Estimating region of each two-pin net is confined within the bounding box. (2) Fractional usage assignment is allowed. The first assumption suggests that we only consider the routing edges inside the bounding box. The second assumption permits breaking the integer usage into fractional values. It keeps in accordance with the objective to evenly distribute the net usage, because fractional values offer more freedom in filling routing demands.

For the runtime consideration, the estimation needs be kept simple and effective. The simplest way of predicting congestion is probabilistic based. For instance, figure 3 illustrates a routing example with four two-pin nets, A, B, C and D. Here we suppose: (1) the edge

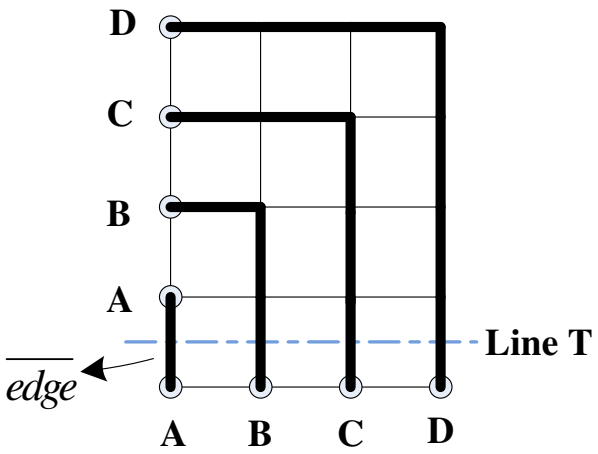

Figure 2.3 Probabilistic estimation.

capacity is 1; (2) estimation goes inside each bounding box; (3) each two-pin net has equivalent probability to pass through the edges along the same row or column. In this case, net D has 1/4 of probability of passing each of the vertical edge along line T. The same rule applies, the total probabilistic usage of the leftmost vertical edge, here we denotes as $\overline{edge}$, will become: $(1 + 1/2 + 1/3 + 1/4)$. Likewise, for the case with N such nets, the probabilistic usage of $\overline{edge}$ will be: $U(\overline{edge}) = (1 + 1/2 + 1/3 + \cdots 1/N)$. The function is also called harmonic function which diverges when N approximates infinity. In other word, the function will generate a large estimated usage for $\overline{edge}$. However, as depicted in figure 3, the routing case, regardless of the value of N, is entirely routable without detour. Hence the desired estimated usage of $\overline{edge}$ is 1. It fully shows the deficiency of the traditional probabilistic usage assignment technique.

FastRoute3.0 utilizes ACE instead to perform the usage assignment. The notation of problem formulation is shown in table 1. Each two-pin net has the usage of 1, and the objective is to assign the usage to global routing edges more evenly. As the example discussed above, the problem is too much usage is piled on some edge, which could be assigned elsewhere. Also, since the usage assignment is applied in a sequential order, the permutation method prominently affects the accuracy. Therefore in sum, there are two sub problems involved: (1) the usage assignment for single two-pin net; (2) the sequential assignment ordering of a group of

nets.

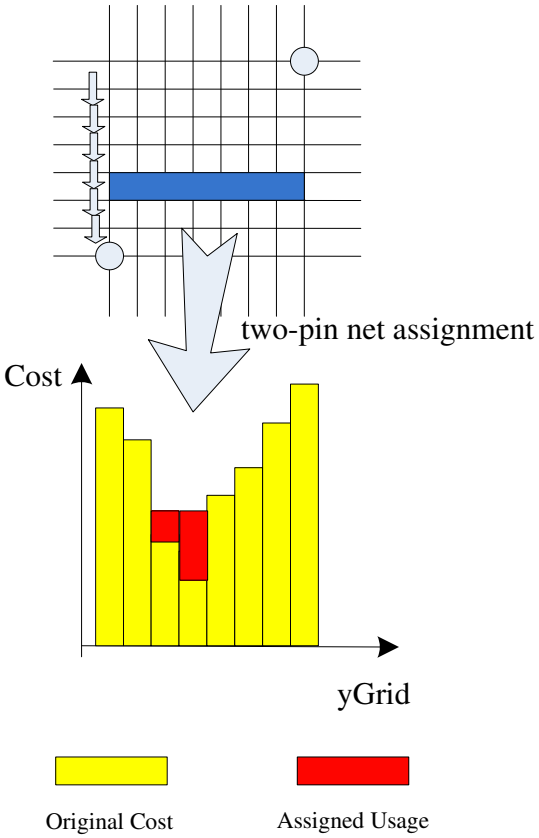

Figure 2.4 Two-pin net usage assignment(vertical case).

Two-pin net assignment Consider the usage assignment of one single two-pin net, the usage ready to be assigned within the bounding box is 1. Without loss of generality, here we discuss the assignment for vertical edges. The same criteria will be applied for horizontal edges. The usage assignment algorithm for vertical edges is shown in figure 5. Each row is processed independently. Inside one row, edges are sorted in a decreasing order according to the value of $cost^V_{i,j}$, which is equal to $p^V_{i,j} + m^V_i - c^V_{i,j}$. m^V_i is the value of maximum edge capacity of row i. The algorithm compares the average potential assigned usage with largest current assigned usage. It iteratively excludes the edge with largest current assigned usage until an even assignment is possible. The time complexity required for processing single two-pin net

Table 2.1 ACE usage assignment notation

N	number of two-pin nets
$BBox_k$	bounding box of net_k
r_k	number of rows inside $BBox_k$
c_k	number of columns inside $BBox_k$
$left_k$	left coordinate of $BBox_k$
$right_k$	right coordinate of $BBox_k$
top_k	top coordinate of $BBox_k$
$bottom_k$	bottom coordinate of $BBox_k$
$c_{i,j}^{V/H}$	capacity of the $edge_{i,j}^{V/H}$
$p_{i,j}^{V/H}$	current assigned usage of $edge_{i,j}^{V/H}$

net_k is $O(r_k c_k \cdot log(c_k))$. Figure 4 illustrates the assignment process.

Net processing order *ACE* chooses to process smaller span nets with higher priority. The net span represents width of bounding box in vertical edge assignment and likewise height of bounding box in horizontal edge assignment. In experiment we discover that nets with larger spans offer more choices to distribute the net usage. Therefore we permutate the net by net span and perform usage assignment for smaller span nets first. Figure 6 shows the detail of the whole *ACE* algorithm.

Now we apply *ACE* to solve the routing example in figure 3. Consider vertical edges along the line T as before. Due to the permutation, the net processing order becomes A→B→C→D. After assigning net A, current assigned usage becomes(1,0,0,0). And we will get (1,1,0,0) after assigning net B. As it goes on, the final assigned usage will be (1,1,1,1). So the estimation won't report any potential congestion, which matches exactly with the analysis.

After the estimation, virtual capacity will be initialized by equation 1.

$$vc_e = rc_e - (max(0, p_e - rc_e)) \quad \forall e \tag{2.1}$$

In the equation, rc denotes real edge capacity, p is the final assigned usage obtained by *ACE*. The new capacity after subtraction is named virtual capacity, which is vc in abbreviation.

Now we analyze the time complexity of *ACE* technique. The ordering of N two-pin nets takes $O(N \cdot log(N))$. For each net, the worst case time complexity is $O(G^2 log(G))$, G is the

Algorithm ACE two-pin net assignment vertical(net_k)

begin
1 **for** $(i = top_k \cdots bottom_k + 1)$
2 $m_i^V = max(c_{i,j}^V), j \in [left_k, right_k]$
3 $\Delta = right_k - left_k + 1$
4 **for** $(j = left_k \cdots right_k)$
5 $cost_{i,j}^V = p_{i,j}^V + m_i^V - c_{i,j}^V$
6 $C_{sum} = \sum cost_{i,j}^V$ $(j \in [left_k, right_k])$
7 Sort $cost_{i,j}^V (j \in [left_k, right_k])$ by decreasing order
8 Copy sorted edge index into queue Q
9 **for** $(t = 1 \cdots \Delta)$
0 **if** $\frac{1+C_{sum}}{\Delta - t + 1} > cost_{i,Q(t)}^V$
1 **for** $(n = t \cdots \Delta)$
2 $p_{i,Q(n)}^V = \frac{1+C_{sum}}{\Delta - t + 1} - m_i^V + c_{i,Q(n)}^V$
3 break out of the second for loop
4 **else**
5 $C_{sum} = C_{sum} - cost_{i,Q(t)}^V$
6 **end for**
7 **end for**
end

Figure 2.5 The ACE two-pin net assignment algorithm for vertical edges

maximum number of horizontal and vertical grids. Hence, in general, the overall worst case time complexity is $O(N \cdot log(N) + NG^2 \cdot log(G))$. But the bounding box of a two-pin net is generally small, therefore on average, *ACE* accounts for nearly 20% of total runtime.

Virtual Capacity Update

After the virtual capacity initialization, FastRoute3.0 substitutes virtual capacity for the real edge capacity to guide maze routing. But as the ripup and reroute proceeds, especially after several iterations, the initial virtual capacity value becomes less effective. Sometimes it even misleads the router. One reason for causing this phenomena is that the congestion estimation is performed within the bounding box. However, that assumption is not supported by maze routing, which is very likely to generate a lot of detour during rip up and reroute iterations.

Algorithm ACE usage assignment()

begin
1 $p^V_{i,j} = 0 \quad \forall i, j$
2 $p^H_{i,j} = 0 \quad \forall i, j$

3 Sort two-pin nets by BBox width with increasing order
4 Copy sorted nets into queue Q^V
5 **for** $(t = 1 \cdots N)$
6 ACE two-pin net assignment vertical($Q^V(t)$)
7 **end for**

8 two-pin nets by BBox height with increasing order
9 Copy sorted nets into queue Q^H
0 **for** $(t = 1 \cdots N)$
1 ACE two-pin net assignment horizontal($Q^H(t)$)
2 **end for**
end

Figure 2.6 The ACE usage assignment algorithm

Therefore, to fix this inconsistency, the virtual capacity needs be updated dynamically. In FastRoute3.0, it is updated at the end of each maze routing iteration.

The update method is presented in equation 2 and 3. Existing overflow o_e is calculated as the difference between edge usage u_e and real edge capacity rc_e. Virtual capacity will be monotonically decreased for the edges that are consistently congested.

However, we could notice that the virtual capacity reduction procedure is irreversible and the capacity is continually lost. So it's highly possible that more and more extra wirelenth will be created by the edges with very small virtual capacity, even though some edges are not congested at all. As a result, we apply virtual capacity increase if the edge usage is below real edge capacity(o_e is less than 0). In equation 3 the augmenting factor F is set to be 0.85 by experiment.

$$o_e = u_e - rc_e \quad \forall e \tag{2.2}$$

$$vc_e = \begin{cases} vc_e - o_e & if \quad o_e \geq 0 \\ vc_e - F \times o_e & if \quad o_e < 0 \end{cases} \tag{2.3}$$

Virtual capacity technique is effective in overflow reduction. Table 2 compares the two modes(with and without virtual capacity) on seven routable ISPD07 global routing benchmarks.

Table 2.2 Comparison of different modes on 7 routable 3D version ISPD07 benchmarks

name	mode1			mode2		
	wlen (e5)	cpu (sec)	OF	wlen (e5)	cpu (sec)	OF
adaptec1	55.1	302	0	/	1800	42
adaptec2	53.6	38	0	/	1800	312
adaptec3	133	249	0	132	639	0
adaptec4	122	54	0	122	77	0
adaptec5	161	682	0	/	1800	436
newblue1	48.2	316	0	/	1800	1348
newblue2	76.3	24	0	76.2	36	0

In the table, mode 1 utilizes virtual capacity in maze routing and mode 2 is traditional maze routing, which switches off virtual capacity technique. Maze routing with virtual capacity apparently shows much better performance in terms of congestion reduction and runtime. Traditional maze routing could only finish routing 3 benchmarks. Note that the runtime is limited to be within 30 minutes.

Quality Improvement Techniques

In this section, we will present another two simple but effective techniques: (1) via aware maze routing; (2) adaptive maze function.

Via aware maze routing

In FastRoute3.0, almost about half of the vias are generated by routing bends. To suppress the number of vias, via cost is incorporated into the maze routing cost function. During the dijkstra expansion, we record the predecessor of current grid position. If the new expansion causes any routing bends, via cost is added to the maze routing cost.

$$cost_e = cost_e + viacost \tag{2.4}$$

Table 2.3 Comparison of via aware maze routing and via ignored maze routing on 7 routable 3D version ISPD07 benchmarks

name	via ignored maze routing			via aware maze routing		
	#via	cpu (sec)	OF	#via	cpu (sec)	OF
adaptec1	2007K	293	0	1843K	302	0
adaptec2	2033K	35	0	1989K	38	0
adaptec3	3848K	238	0	3600K	249	0
adaptec4	3306K	51	0	3287K	54	0
adaptec5	5667K	672	0	5489K	682	0
newblue1	2425K	312	0	2314K	316	0
newblue2	3009K	21	0	2992K	24	0

In table 3 we compare the via aware maze routing and via ignored maze routing on seven routable ISPD07 3D benchmarks. We could observe that the extra via cost could effectively remove over 3% 3D vias with 2% increase of runtime.

Adaptive Maze Function

The adaptive maze routing cost function is presented in equation 5. In the function. k is the coefficient controlling the function curve slope when u_e is below c_e. k is adaptively adjusted in different maze routing phases. In the initial phase, the k is set small to preserve good wirelegnth. Normally in the first few iterations, many nets need ripup and reroute. If a large k coefficient is applied, excessive routing wires would be rerouted with huge detour. While in the final stage of maze routing, the cost function curve is made steep to aggressively drive down the residual overflow. There are three other coefficients in the function: M is the cost when u_e is equal to c_e. S determines the slope when u_e is over c_e. H is the cost height which is increased each maze routing iteration. The parameters are experimentally determined.

$$cost_e = \begin{cases} 1 + H/(1 + exp(-k(u_e - c_e))) & if \quad 0 < u_e \leq c_e \\ 1 + M + S \times (u_e - c_e) & if \quad u_e > c_e \end{cases} \tag{2.5}$$

Results and Discussion

FastRoute3.0 is implemented in C, and all the experiments are performed on one 2.4Ghz Intel processor with 4GB of RAM. FLUTE [26] is utilized to generate RSMT. We demonstrate FastRoute3.0's performance by running three benchmark suites: ISPD98 benchmarks [20], 3D version of ISPD07 global routing contest benchmarks [28] and 3D version of ISPD08 global routing contest benchmarks [29]. The benchmark statistics are shown in table 4.

ISPD98 benchmarks

Table 5 shows the FastRoute3.0's performance for ISPD98 benchmarks. We make comparison with recently published academic global routers: NTHU-R, BoxRouter 2.0, Archer, FGR, FastRoute2.0 and BoxRouter. The results are quoted from [12], [10], [9], [11], [24] and [6] respectively. First, the result shows that FastRoute3.0 is able to route through all the benchmarks without any overflow. Second, FastRoute3.0 achieves good runtime. It can finish routing all 10 benchmarks within 15 seconds on our platform. Among all quoted global routers, FastRoute2.0 achieves fastest runtime. But it fails to generate congestion free solutions for ibm01, ibm04 and ibm09. Third, the total wirelength is comparable with Archer, NTHU-R, FGR and BoxRouter 2.0, which is 2.26% and 1.96% better than FastRoute2.0 and BoxRouter.

3D version of ISPD07 benchmarks

Table 6 shows routing results on 3D version of ISPD07 global routing contest benchmarks. As shown in table 4, they are much harder in routing complexity and larger in grid size. We compare FastRoute3.0 with recently published global routers: FGR [11], MaizeRouter [27], BoxRouter 2.0 [10], FastRoute2.0 [24], Archer [9] and NTHU-R [12]. The first three routers

Table 2.4 Experimental benchmarks statistics

Name	Grids	#Nets	#Routed Nets	Max Deg	Avg Deg
ibm01	64×64	11.5k	9.1k	37	3.8
ibm02	80×64	18.4k	14.3k	126	4.4
ibm03	80×64	21.6k	15.3k	49	3.6
ibm04	96×64	26.2k	19.7k	41	3.4
ibm06	128×64	33.4k	25.8k	34	3.8
ibm07	192×64	44.4k	34.4k	22	3.8
ibm08	192×64	47.9k	35.2k	65	4.3
ibm09	256×64	50.4k	39.6k	38	3.8
ibm10	256×64	64.2k	49.5k	32	4.2
adaptec1	324×324	219k	177k	340	4.2
adaptec2	424×424	260k	208k	153	3.9
adaptec3	774×779	466k	368k	82	4.0
adaptec4	774×779	515k	401k	171	3.7
adaptec5	465×468	867k	548k	121	4.1
newblue1	399×399	332k	271k	74	3.5
newblue2	557×463	463k	374k	116	3.6
newblue3	973×1256	552k	442k	141	3.2
bigblue1	227×227	283k	197k	74	4.1
bigblue2	468×471	577k	429k	260	3.5
bigblue3	555×557	1.12M	666k	91	3.4
bigblue4	403×405	2.23M	1.13M	129	3.7
newblue4	455×458	636k	531k	152	3.6
newblue5	637×640	1.26M	892k	258	4.1
newblue6	463×464	1.29M	835k	123	3.8
newblue7	488×490	2.64M	1.65M	113	3.6

Table 2.5 Comparison of FastRoute3.0, and published global routers on ISPD98 Benchmarks.

name	FastRoute3.0			NTHU-R [12]			BoxRouter 2.0 [10]			FGR [11]			Archer [9]			FastRoute2.0 [24]			BoxRouter [6]		
	OF	wlen	cpu (sec)	OF	wlen	cpu (sec)	OF	wlen	cpu (sec)	OF	wlen	cpu (sec)	OF	wlen	cpu (sec)	OF	wlen	cpu (sec)	OF	wlen	cpu (sec)
ibm01	0	64221	0.64	0	63321	4.17	0	62659	33	0	63332	10	0	64389	11	31	68489	0.72	102	65588	8
ibm02	0	172223	0.85	0	170531	7.44	0	171110	36	0	168918	13	0	171805	25	0	178868	0.93	33	178759	34
ibm03	0	146753	0.49	0	146551	5.86	0	146634	18	0	146412	5	0	146770	10	0	150393	0.60	0	151299	17
ibm04	0	170146	2.70	0	168262	13.61	0	167275	116	0	167101	29	0	169977	24	64	175037	1.88	309	173289	24
ibm06	0	279471	1.15	0	278617	12.75	0	277913	47	0	277608	6	0	278841	23	0	284935	1.36	0	282325	33
ibm07	0	369023	1.68	0	366288	15.89	0	365790	86	0	366180	18	0	370143	25	0	375185	1.60	53	378876	51
ibm08	0	405935	1.82	0	405169	13.17	0	405634	90	0	404714	18	0	404530	42	0	411703	2.36	0	415025	93
ibm09	0	414913	1.67	0	415464	11.59	0	413862	273	0	413053	20	0	414223	37	3	424949	1.92	0	418615	64
ibm10	0	582838	3.61	0	580793	33.72	0	590141	352	0	578795	92	0	583805	45	0	595622	2.79	0	593186	95
Total	0	2606K	14.61	0	2595K	118.20	0	2601K	1051	0	2585K	211	0	2604K	242	98	2665K	14.16	497	2657K	419
Norm	/	1	1	/	0.996	8.09	/	0.998	72.94	/	0.992	14.44	/	1	16.56	/	1.023	0.97	/	1.020	28.68

Figure 2.7 FastRoute3.0 overflow reduction during maze routing iteration.

are the winners of the 2007 ISPD global routing contest [28]. Speaking of final overflow, FastRoute3.0 is able to complete 7 benchmarks without any congestion. Noticeably, FastRoute3.0 successfully route through newblue1, which is not routable by any of the referred routers. For the unroutable one, newblue3, FastRoute3.0 produces the solution with lowest overflow. Secondly, in order to show FastRoute3.0's strong point on runtime, we run FGR 1.1 on our platform. The total runtime added together is around one hour, which is 64× faster than FGR 1.1. The runtime of Archer is also reported. From the literally quoted results [9], FastRoute3.0 is 9× faster. Note that Archer is performed on an Intel Xeon 3.60Ghz processor. Considering the wirelength, the reported results are calculated by the ISPD07 cost(segment wirelength plus three times of via number). FastRoute3.0 is comparable with all the winners of the contest, but it is much better than FastRoute2.0 with over 50% improvement. We also investigate the convergence of our router. Figure 8 shows that the total overflow of adaptec1, adaptec2, and adaptec5 goes down in logarithmical order and the required maze iteration is only around 20.

Table 2.6 Comparison of FastRoute3.0, and published global routers on 3D version of ISPD07 global routing contest benchmarks

name	FastRoute3.0			FGR 1.1 [11]			Archer [9]			BoxRouter2.0 [10]		MaizeRouter [27]		FastRoute2.0 [24]	
	OF	wlen (e5)	cpu (min)	OF	wlen (e5)	cpu (min)	OF	wlen (e5)	cpu (min)	OF	wlen (e5)	OF	wlen (e5)	OF	wlen (e5)
adaptec1	0	92	5	0	88.5	345	0	114	87	0	92	0	100	122	249
adaptec2	0	93.4	1	0	90	32	0	113	23	0	94	0	98	500	244
adaptec3	0	205	4	0	200	200	0	244	51	0	207	0	214	0	523
adaptec4	0	188	1	0	179	29	0	222	12	0	186	0	194	0	469
adaptec5	0	271	11	0	260	748	0	334	248	0	270	0	305	9680	708
newblue1	0	94.5	5	314	94	1083	494	116	50	400	92.9	1348	102	1934	248
newblue2	0	136	1	0	129	9	0	167	7	0	135	0	140	0	380
newblue3	31634	182	36	45454	164	1513	31928	199	163	38958	172	32588	184	34236	443

Table 2.7 Comparison of FastRoute3.0, and ISPD08 global routing contest results

name	FR3.0			NTHU-R [23]			NTUgr [23]			FR3.0c[23]			BoxRouter [23]			FGR [23]		
	OF	wlen (e5)	cpu (min)	OF	wlen (e5)	cpu (min)	OF	wlen (e5)	cpu (min)	OF	wlen (e5)	cpu (min)	OF	wlen (e5)	cpu (min)	OF	wlen (e5)	cpu (min)
adaptec1	0	55.2	5	0	53.5	8	0	56.1	5	0	55.5	2	0	53.8	20	0	54.1	35
adaptec2	0	53.7	1	0	52.3	2	0	53.4	1	0	53.1	1	0	52.7	3	0	52.6	14
adaptec3	0	133	4	0	131	8	0	134	5	0	133	2	0	132	27	0	132	55
adaptec4	0	123	1	0	122	2	0	123	2	0	122	1	0	122	7	0	122	17
adaptec5	0	161	11	0	156	17	0	159	17	0	161	5	0	157	31	0	157	110
newblue1	0	48.2	5	0	46.5	5	6	50.3	1150	76	49	13	44	47	1241	8	46.8	1412
newblue2	0	76.3	1	0	75.7	1	0	77.4	1	0	76.2	1	0	75.9	2	0	75.8	4
newblue3	31634	110	36	31454	106	129	31106	180	809	31650	109	31	32404	109	1377	34850	106	1427
bigblue1	0	59.5	7	0	56.3	10	0	57.8	14	0	58.3	4	0	57	19	0	57.3	70
bigblue2	0	98.8	16	0	90.6	10	0	97.2	264	142	98.2	11	0	90.4	39	0	91.4	238
bigblue3	0	132	2	0	131	4	0	136	5	0	132	3	0	131	6	0	132	88
bigblue4	156	244	35	182	231	126	188	243	413	206	243	41	472	232	877	414	232	1425
newblue4	154	137	17	152	130	67	142	144	1118	226	136	10	200	129	1304	262	130	1420
newblue5	0	240	11	0	232	14	0	246	28	0	241	5	0	233	28	0	233	166
newblue6	0	186	10	0	177	14	0	186	16	0	187	4	0	180	30	0	180	103
newblue7	108	361	160	68	354	141	310	372	1446	588	359	190	208	351	1412	1458	350	1434

3D version of ISPD08 benchmarks

In the ISPD 2008 global routing contest, 8 benchmarks are newly released. We evaluate FastRoute3.0's performance on the new benchmarks in table 7. The comparison is made with top 5 contest winners: NTHU-R, NTUgr, FastRoute3.0c [1], BoxRouter 2.0 and FGR. The contest results are obtained by running each submitted binary on a machine with up to four 2.8Ghz AMD processors[23]. First of all, in terms of overflow, FastRoute3.0 finishes routing 12 out of 16 benchmarks, which is the same as the best known results. Second, the total runtime added together is the second smallest. Although FastRoute3.0c achieves fastest runtime, it fails to route through newblue1 and bigblue2. Besides, NTUgr and FastRoute3.0c utilize multi-core programming, hence their single thread runtime would be slower. The runtime comparison may not be very accurate as the contest binaries are still not publicly available. Here we only want to demonstrate the runtime advantage of our router, which is at least comparable with the contest winners. Third, the ISPD08 wirelength cost(segment wirelength plus number of via) is on the same level with the others. It indicates that FastRoute3.0 doesn't sacrifice wirelength for achieving good runtime.

Conclusion

In this paper, we have proposed FastRoute3.0, a fast and high quality global router with special emphasis on overflow reduction. The newly introduced technique is virtual capacity, which is used to guide the global router in maze routing stage out of local optimal solutions. FastRoute3.0 generates high quality solutions for ISPD98, ,ISPD07 and ISPD08 benchmark suites. But due to the fast growing problem size and degree of routing complexity, our future work will focus on two aspects. First, we will continue to improve FastRoute 3.0's routability and runtime. Second, we will try to apply it in earlier physical design stage to produce routing friendly placement.

[1] FastRoute3.0c is our contest version

CHAPTER 3. CROP: Fast and Effective Placement Refinement for Routability

A paper submitted to IEEE Transactions on Computer-Aided Design

Yanheng Zhang and Chris Chu

Abstract

Routability is becoming more and more challenging as the design feature continues to decrease. Previous routability-driven placement techniques are often tightly coupled with the underlying placers and they cannot be easily integrated into various placement tools.

In this paper, we propose CROP (Congestion Refinement of Placement) for improving mixed-size placement solutions. CROP is independent of any placer. It is imported with any legalized placement solution and relocates the modules to improve routability without significantly disturbing the original placement solution.

CROP interleaves a congestion-driven module shifting technique and a congestion-driven detailed placement technique. Basically the shifting technique targets at better allocating the routing resources. Shifting in each direction can be formulated as a linear program (LP) for resizing each cell in global routing grid (i.e., G-Cell). Instead of solving the computationally expensive LP, we discover that the LP formulation could be relaxed and solved by a very efficient longest-path computation. Then the congestion-driven detailed placement technique is proposed to better distribute the routing demands. Congestion reduction is realized by weighting the half-perimeter wirelength (HPWL) with congestion factor during detailed placement.

Our tool is capable of handling any mixed-size placement benchmark with movable and/or fixed macro blocks. In order to better analyze its performance, two sets of benchmarks: ISPD-

GR (ISPD05/06 derived global routing benchmkars) and ISPD-DR (ISPD05/06 derieved detailed routing benchmarks) are developed. The experimental results show that CROP effectively alleviates the congestion for unroutable placement solutions in short runtime for different placers.

Introduction

The success of routing is critical in VLSI design flow. With the ever decreasing feature size, the routability issue has become more and more complicated. Nowadays, the mixed-size SOC contains up to millions of standard cells and thousands of big macros in one single design. The existence of big macros and large problem size make the routability issue more and more challenging.

In traditional design flow, routing and placement are treated as independent stages. In the placement stage, typically, HPWL is set as primary objective for optimization. Nevertheless, the HPWL optimization during the placement stage may lead to a hard-to-route or even unroutable solution. It is desirable to integrate routing consideration during placement. Actually placement is an early and more flexible stage for improving routability. The shifting and relocating of modules could effectively produce much higher QoR for various design objectives including routability. Therefore, congestion-driven placement techniques received much attention.

There have been many works proposed for routability-driven placement. In general, previous techniques could be categorized into four groups. The first group incorporates routability components into placement optimizing objective. Spindler and Johannes [30] proposed RUDY congestion estimation technique and modified the density term to contain both the routing density and module density. In [31], Jiang et al. applied Lagrangian relaxation to relax the routability constraints. Similarly, Tosta et al. [32] integrated the wire density term into the analytical placement framework. The second group applies implicit or explicit White Space Allocation (WSA) technique inside or after the placement flow. Yang et al. [33] proposed three WSA methods and integrated one of them in the detailed placement flow of Dragon. mPL-R

with WSA[34] distributed the white space by adjusting the cut-lines of hierarchically sliced placement based on the available white space and congestion. In [35], the authors proposed inflating the cells inside the congested region, which is an implicit manner for allocating white space. The third group guides placement by global routing. IPR[36] integrated FastRoute2.0 [8] into FastPlace[37] and performed full global routing to guide the placement flow. The fourth group mixes some of the above three features. For instance, ROOSTER[38] proposed to optimize RSMT in their global placement objective and apply WSA in their detailed placement flow.

In this work, we propose a fast and effective mixed-size placement refinement tool called CROP for routability improvement. CROP interleaves congestion-driven module shifting technique and congestion-driven detailed placement technique. Both techniques are guided by congestion information obtained by global routing. The congestion-driven shifting technique targets at better allocating the routing resources. It is achieved by adjusting the boundary of each G-Cell and shifting the modules according to the new G-Cell shape. Figure 3.1 illustrates the basic idea. In the figure, the G-Cell has insufficient capacity for accomodating the routing demands. Since the global routing capacity is linearly related to the length of the G-Cell boundary. If the G-Cell is enlarged proportional to the demand of routing tracks, theoretically no routing overflow would occur. We will show that the resizing of G-Cell can be formulated as two linear programs (LPs) for vertical and horizontal shifting respectively. Instead of solving the computational expensive LP, we relax the LP and solve it by an efficient longest-path computation, which is the major factor contributing to our fast runtime when shifting the modules. After performing module shifting, we will legalize the placement solution. Then we will apply our second technique, the congestion-driven detailed placement (DP) to probe better routability. It aims at better distributing routing demands. Congestion reduction is realized by weighting the HPWL with a congestion coefficient during detailed placement. The Shifting-Legalization-DP procedure forms one iteration of refinement. We will call the refinement repeatedly until the solution is good enough. Practically only a small number of iterations (usually 2-3) is sufficient to achieve good routability.

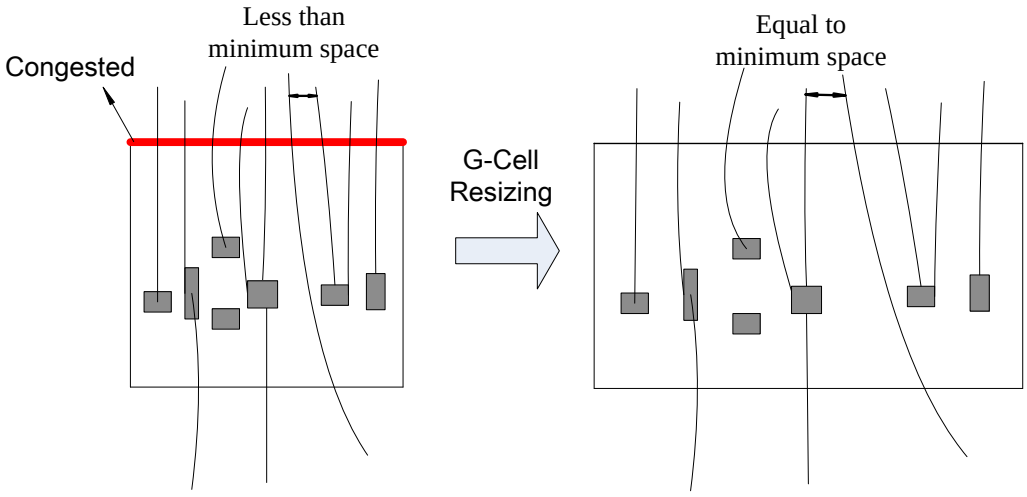

Figure 3.1 Basic idea of congestion-driven module shifting.

CROP is a fast and effective refinement tool for mixed-size placement solution with several nice properties. First, CROP is independent of any placer. Previous congestion-driven techniques are very tightly coupled with the underlying placer. Their proposed methods cannot be easily integrated into various placement tools. Second, CROP shows good performance in improving the routability. The difference of our model of module shifting from previous works (e.g., mPL-R+WSA[34]) is that ours is more refined. Instead of shifting the cut-lines of a hierarchically sliced layout, CROP shifts the boundary of each G-Cell. Moreover, previous techniques usually lump vertical and horizontal congestion together. Yet our congestion shifting model differentiates the vertical and horizontal directions. Third, CROP runs very fast. For instance, the design with 800k modules and 800k nets (adaptec5) takes less than 10 minutes to execute.

In summary, our technical contributions include the introduction of the following:

- A placement routability refinement flow which is independent of any placer and router
- A more refined and directional module shifting model
- A longest-path computation method aiding fast runtime of module shifting
- A congestion-driven global swap technique by weighting HPWL with congestion.

We apply CROP to refine placement solutions obtained from various placers: FastPlace 3.1[39], NTUplace3[40], mPL6[41] and R-NTUplace3[31]. We set up ISPD-GR and ISPD-DR benchmarks to verify its performance. The results reveal that CROP effectively reduces congestion within a very short runtime.

The rest of paper is organized as follows: Section II provides preliminaries on global routing, the general flow of CROP and the methodology for performing congestion estimation. In Section III, we introduce the details of congestion-driven module shifting technique. Section IV explains congestion-driven detailed placement. In Section V, we make comparison for results on ISPD-GR and ISPD-DR benchmarks and conclusion will be made in Section IV.

Materials and Methods

Preliminaries

In this section we will introduce some terminology and present the overview of CROP. We will also talk about the congestion estimation method that guides CROP.

Motivation

The placement region is partitioned into a set of G-Cells to perform global routing. The G-Cells are illustrated in Figure 3.2. The global routing will be performed across the boundaries between adjacent G-Cells. In the global routing grid graph, each G-Cell will be represented by a node and each G-Cell boundary will be represented by an edge between two nodes, which is referred to as global routing edge. If the usage U_e is over capacity C_e for any edge e, the overflow is calculated as $O_e = U_e - C_e$. If there is no congested edge, then the design is routable in global routing stage.

To improve the routability, we could adopt two methods in general. The first method is to allocate more routing resources, or in other words, to increase global routing capacities. As is notable, the global routing edge capacity is proportional to the length of the G-Cell boundary. It can be easily calculated by dividing the length of boundary over minimum pitch

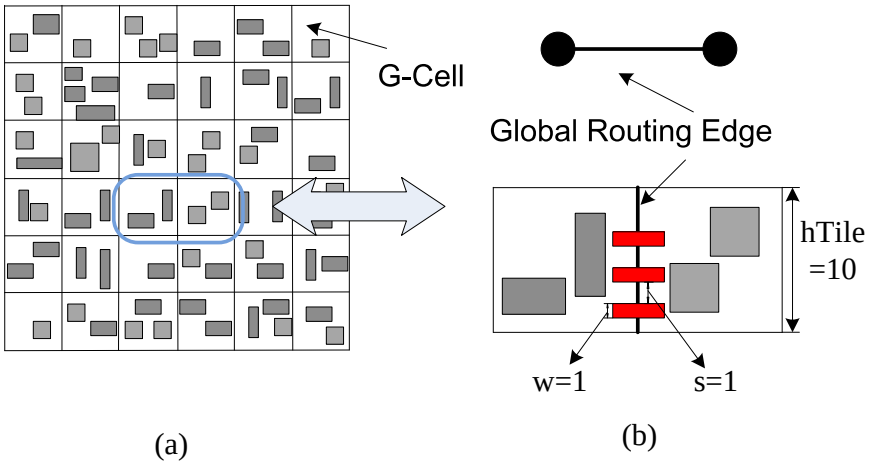

Figure 3.2 An illustration of G-Cells and global routing across G-Cell boundary.

size. For instance, in Figure 3.2, the highlighted G-Cell boundary has global routing capacity of 5 (boundary length is 10, minimum pitch is 2.)[1] Due to the linear relationship between G-Cell boundary length and capacity, it motivates our congestion-driven module shifting technique by reshaping each G-Cell for better allocating routing resources (e.g., adjust the boundary length for each G-Cell). The second method is to reduce and relocate routing demands. After module shifting, a step to regain the degraded wirelength is necessary to prevent extensive elongation of nets. We thus proposed the congestion-driven detailed placement to compensate the wirelength loss after the module shifting stage. Besides reclaiming the wirelength, we further move routing demands away from congested regions by applying congestion aware global swap. Hence, CROP considers both resource allocating and demand reduction, and the details will be fully discussed in Section III and Section IV respectively.

CROP Flow

The flow of CROP is illustrated in Figure 4.3. The imported design should have been placed and legalized. The first major step is congestion-drive module shifting. As discussed in last paragraph, it is applied to adjust the position of modules for better resource allocation.

[1]Capacity calculation can be complicated in real design and varies depending on many different factors. Here is our simple capacity model for motivation

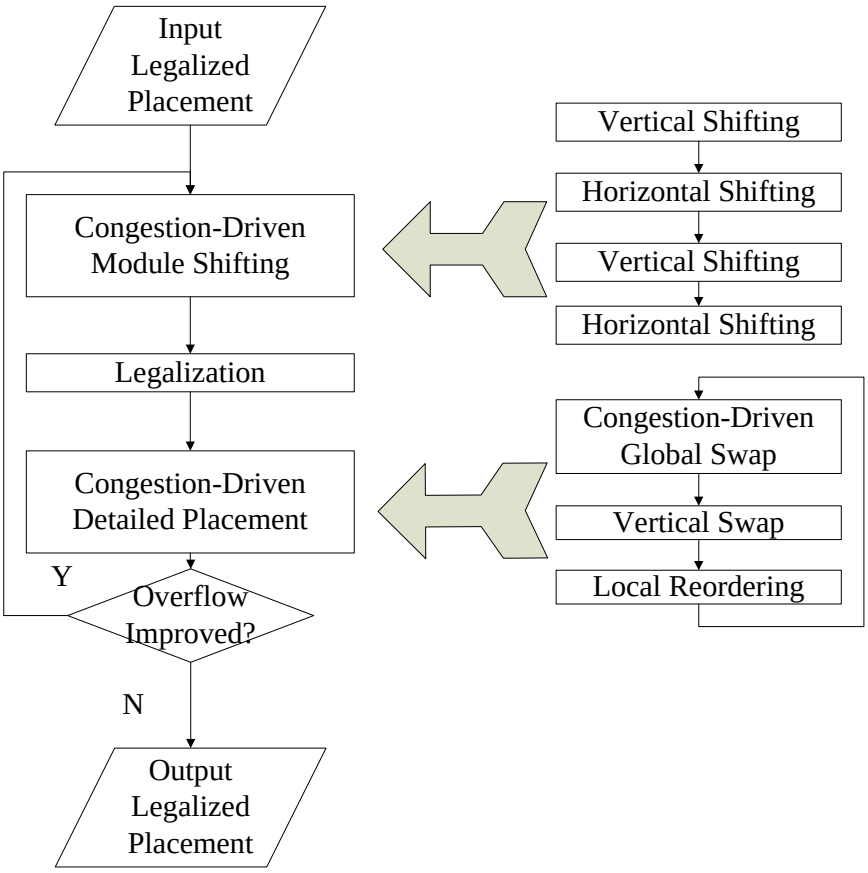

Figure 3.3 Algorithm flow.

The goal is achieved by resizing G-Cells. The congestion-driven module shifting contains four substeps. Each substep is performed either on X or Y direction to optimize the solution from different direction iteratively. When one direction is processed, let's say X, we will fix the module postions in Y direction. After a number of rounds of module shifting (2 rounds in Figure 4.3), the post-shifting placement solution will be legalized. Then congestion-driven detailed placement will be called to compensate the wirelength loss and further improve the routability. The mentioned procedure will be repeatedly called until the solution gets stable. Normally it only takes two to three iterations for obtaining considerable routability improvement.

Congestion Estimation

To improve routability, we need fast and accurate congestion information of the placement solution. It is a very important component of congestion-driven placement. There are various algorithms proposed for estimating congestion. Previous congestion evaluation could be roughly grouped into two categories, bounding box based or global routing based. Bounding box based approach seeks to estimate the routing usage in a probabilistic manner inside the bounding box. While the global routing based method implement an efficient global router to test the routability. Although runs faster, the bounding box method is an inaccurate way for measuring congestion since no routing structure is actually constructed. The global routing based estimation, on the other hand, build tentative routing topology and implement simple pattern routing (e.g., L/Z shaped pattern routing). Hence the global routing based estimation is also referred to as topology based estimation.

In this paper, we apply coarse global routing result as a congestion reporter. In addition to the simple L/Z pattern routing, we incorporate a fast and more accurate 3-bend routing. The technique was proposed in FastRoute 4.0[42]. It has been shown in [42] that the time complexity of 3-bend routing is $O(mn)$ for a net spanning a region of $m \times n$ G-Cells, which is as fast as Z routing. Besides the fast computing time, it is more flexible in choosing routing paths with the ability for making necessary detours over congested areas. The routing output will be closer to the final routing solution when iterative rip-up and reroute is adopted.

In the flow of our tool, both congestion-driven module shifting and congestion-driven detailed placement are guided by global routing. It is applied before each round of vertical shifting, horizontal shifting and congestion-driven global swap.

Congestion-Driven Module Shifting

In this section, we will discuss the congestion-driven module shifting method for moving standard cells. In order to shift the modules with congestion awareness, we first shift the boundary of each G-Cell. The module positions are updated based on the new G-Cell shape and location. In the following subsections, we first formulate LPs of one-dimensional shifting for resizing each G-Cell in X or Y direction respectively. Then we show that the LPs can be relaxed and solved by a more efficient longest-path computation. Module updating method and our approach to handle flexible or fixed big macros will also be discussed in details.

Resizing G-Cells by Linear Programming

We will formulate linear program for resizing G-Cells to accomodate routing usage. To facilitate the formulation, let's first assume each module (M_k) is a standard cell (with smaller area than G-Cell) for the time being.

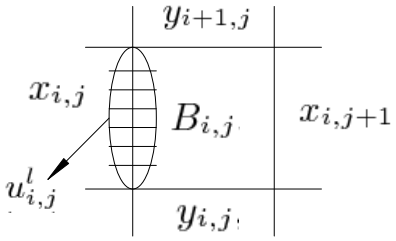

Figure 3.4 Notation for the LP formulation.

We first partition the placement region into $m \times n$ G-Cells. Let $B_{i,j}$ represents each G-Cell, where i ($i \in \{1, ..., m\}$) denotes the row and j ($j \in \{1, ..., n\}$) denotes the column. We introduce coordinate variables for each of the G-Cell boundary. For $B_{i,j}$, let $x_{i,j}$ and $x_{i,j+1}$ denote the

x-coordinate for **left boundary** and **right boundary** respectively. And likewise $y_{i,j}$ and $y_{i+1,j}$ denote the y-coordinate for **bottom boundary** and **top boundary** respectively. There are $m \times (n+1)$ x-variables and $(m+1) \times n$ y-variables. We use $u^l_{i,j}$, $u^r_{i,j}$, $u^b_{i,j}$ and $u^t_{i,j}$ to represent the global routing usage across left, right, bottom and top boundary for $B_{i,j}$. We also introduce H, W, $hTile$ and $wTile$ to denote height of placement region, width of placement region, original height of G-Cell and original width of G-Cell.

Without loss of generality, we only consider the horizontal shifting of vertical boundaries. Similar equations can be derived for the vertical shifting case.

$$\textbf{max : } \sigma$$

s.t.

$$x_{i,j+1} - x_{i,j} \geq \sigma \times MAX(f^{-1}(u^b_{i,j}), f^{-1}(u^t_{i,j})) \qquad \forall i, j \tag{3.1}$$

$$0 \leq \sigma \leq 1 \tag{3.2}$$

$$x_{i,j+1} - x_{i,j} \geq \frac{\sum_{k \in B_{i,j}} area(M_k)}{hTile} \qquad \forall i, j \tag{3.3}$$

$$|x_{i,j} - x_{i+1,j}| \leq C \qquad \forall i, j \tag{3.4}$$

$$0 \leq x_{i,1} \qquad \forall i \tag{3.5}$$

$$x_{i,n+1} \leq W \qquad \forall i \tag{3.6}$$

Then we will explain each of the constraints.

1. *Routability Constraints* (Equations 1 and 2)

   As we mentioned in Section 2, routing capacity is proportional to the length of G-Cell boundary. Ideally, if each G-Cell is sufficiently large, there would be no congestion because each G-Cell boundary is capable of holding the crossing routing wires. In the routability constraints, $x_{i,j+1} - x_{i,j}$ represents the width of $B_{i,j}$. On the right hand side, $f^{-1}(u)$ is the inverse function of $f(l)$, which maps G-Cell boundary length l to routing capacity.

So $f^{-1}(u)$ is the function translating the given usage to sufficient length of boundary. In particular, $f(l)$ is given in Equation 7.

$$f(l) = gB \times (\frac{l}{p} \times gL + \frac{l}{p} \times (layer - 1)) \tag{3.7}$$

p is wire pitch (width plus spacing), $layer$ represents number of 3D routing layers for vertical direction. gB is the guard band coefficient and gL is the ground layer reduction coefficient. Our Length-Capacity model is borrowed from the ISPD07/08 Global Routing Contest[28] [29].

However, if the placement solution is very congested, the routability constraints may be too hard to satisfy. In other words, the formulated LP might be infeasible. Therefore we place a relaxing variable σ in the routability constraints. σ here can be viewed as a scaling factor over original constraint. The value of σ is bounded between 0 and 1. When σ is 1, we do not relax the constraints. With smaller σ value, the routability constraints become less restrictive.

2. *G-Cell Area Constraints* (Equation 3)

   These constraints ensure that each G-Cell has enough space to hold the modules inside. Otherwise, it would create huge overlaps between modules when the non-congested G-Cells shrink excessively. In the formulation, since we consider shifting in X direction, the height of G-Cell is fixed.

3. *Movement Constraints* (Equation 4)

   The input design has already been a legalized placement solution. It is necessary not to disturb the original placement too much. Hence we introduce movement constraints to restrict the shifting between adjacent G-Cell boundaries. In the equations, C is a constant which represents the degree of flexibility of moving adjacent G-Cell boundaries. In experiment we set C to be $0.5 \times wTile$. Because of the absolute sign, it can be expanded as follows,

$$x_{i+1,j} - x_{i,j} \geq -C \qquad \forall i, j \tag{3.8}$$

$$x_{i,j} - x_{i+1,j} \geq -C \qquad \forall i, j \tag{3.9}$$

4. *Placement Region Constraints* (Equations 5 and 6)

   Finally, these constraints ensure that all the boundaries should be within the placement region. Note that the other constraints (e.g., G-Cell area constraints) implicitly guarantee that $x_{i,j} \leq x_{i,j+1}$ for $j \in \{1, \ldots, n\}$.

Longest Path based Solution

The LP is expensive in terms of solving time. Next we will introduce a technique based on longest path to solve it efficiently.

When we investigate the proposed LP in Section 3.1, we find that if σ is fixed, the LP becomes a feasibility check problem (only has constraints). Since Equations 1,3,8 and 9 are all difference constraints, we propose the following strategy to solve the LP. We use an outer loop which keeps decreasing σ until the LP is feasible. Inside the loop, for a fixed σ, we check feasibility by longest path computation. If infeasible, the longest path solution will suggests how σ should be decreased.

Let's first assume σ is fixed. The feasibility of the constraints can be checked by performing a longest path computation on a directed graph called G-Cell boundary graph (B-graph) $G(V, E)$. Each G-Cell boundary associated with $x_{i,j}$ is represented by a vertex $v_{i,j} \in V$. Each difference constraint in the form $x_d - x_s \geq Q$ is represented by a directed edge $e \in E$ pointing from v_s to v_d with a cost $\|e\|$ of Q. To distinguish different edge types, we name E_r, E_a and E_m for the set of edges incurred by routability constraints, G-Cell area constraints and movement constraints respectively. Figure 3.5 illustrates an example of B-graph with the three types of edges. The longest path distance to $v_{i,j}$ from the vertices associated with the leftmost boundaries of the placement region is the minimum value of $x_{i,j}$ that satisfies Equations 1,3,5,8 and 9. So the feasibility of the constraints can be determined by checking whether $x_{j,n+1} \leq W$ for all i (Equation 6).

We observe that the proposed B-graph contains directed cycles. The directed cycles are caused by movement edges (E_m), which are used to control the disturbance of original place-

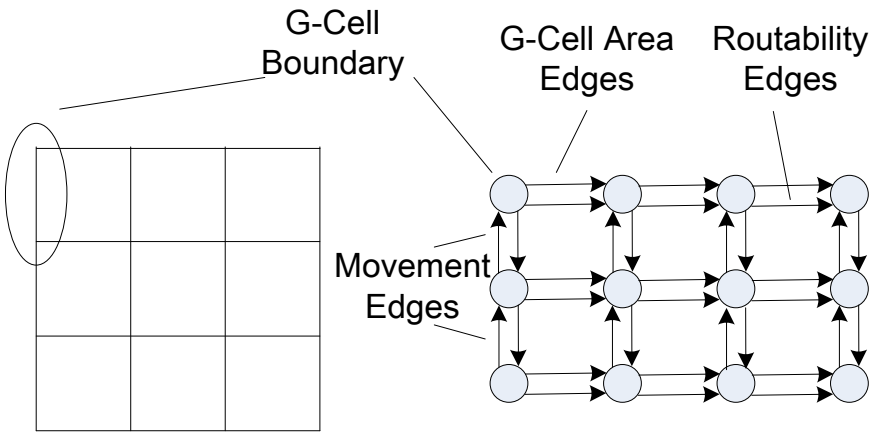

Figure 3.5 Convert the G-Cell boundary into B-graph.

ment. However, as suggested by [3] , it's NP-Complete to find the longest path for a graph with directed cycles. The neat longest path algorithm cannot be applied in this case.

The issue can be resolved by introducing the diagonal edges E_d to replace the hard-to-handle movement edges. Diagonal edges are an alternative method of maintaining original placement solution. Figure 3.6 illustrates the idea. We merge the perpendicular edges ($e1$ and $e2$) and replace $e1$ with the diagonal edge ($e3$). Note that here $e2$ represents the longer one of area edge and the routability edge. The cost of a diagonal edge is the total cost of the perpendicular edges, which is to say, $\|e3\| = \|e1\| + \|e2\|$.

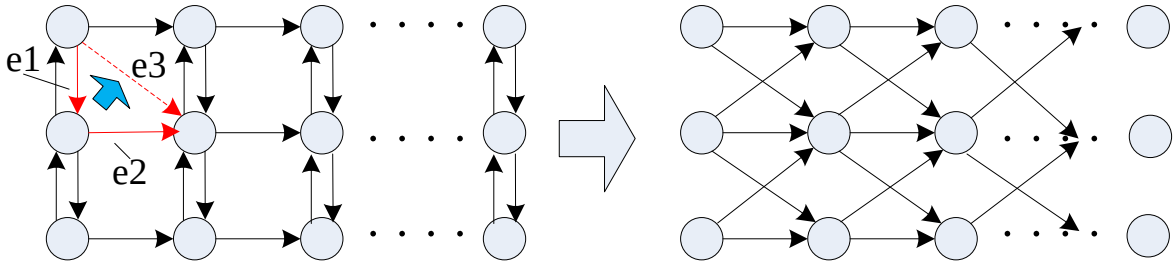

Figure 3.6 Replace movement edges with diagonal edges to facilitate longest path computation.

After replacing the movement edges with diagonal edges, the B-graph has become a DAG. Now we are able to perform longest path computation. It can be done very effecitvely by

one-time scanning of each vertex in B-graph according to the topological order.

Next we will discuss the outer loop for determining the maximized σ to solve the LP. Initially σ is set to 1. If the resulting longest path length $L_p = MAX_{i=1,...,m}(x_{i,n+1})$ is larger than W, we reduce σ to scale the current longest path into placement region, which is the requirment of placement region constraints. Suppose L represents set of edges along the longest path. We define the edges that cannot be scaled as hard edges and those scalable edges as soft edges. For instance, routability edges can alwasy be reduced in magnitude and they are thus soft edges. Area edges are hard edges due to their inscalability. Diagonal edges, on the other hand, can be categorized either into soft edges or hard edge based on which component ($E_r or E_a$) dominates (Since $\|E_d\| = max(\|E_a\|, \|E_r\|) + \|E_m\|$). Hence, we divide the edges along the longest path L into two parts: hard edge $E_h^L = (E_a \cup E_d^h) \cap L$; and soft edge $E_s^L = (E_r \cup E_d^s) \cap L$. $L_h = \sum_{e \in E_h^L} \|e\|$ and $L_s = \sum_{e \in E_s^L} \|e\|$. To scale L_p inside fixed outline W, we have $s \times L_s + L_h = W$. Therefore $s = (W - L_h)/L_s$. Each iteration σ will be scaled by a scaling factor s to configure the soft edges into fixed outline. But we may not be able to compact all paths into fixed outline at one time. First, other paths may still be longer than W even after scaling. Second, current path may not be scaled correspondingly based on s. The reason comes from the fact that the diagnal edges (E_d) have the bound for scaling. They cannot be scaled once the area edge component (E_a) dominates. Hence the scaling in the outer loop will be performed iteratively until all the paths fit into the fixed outline ($L_p = W$). The algorithm terminates in at most m iterations because $x_{i,n+1}$ for at least one more i will become less than or equal to W in each iteration. In practice, it usually takes less than 10 iterations. Figure 3.7 shows our complete algorithm to solve the LP.

We have discussed the algorithm for solving the resizing problem by longest path based solutions. And our algorithm assumes $x_{i,1} = 0 \quad \forall\ i$. However, the potential problem for such assumption is it will cause the modules to be shifted to the left bound. Please refer to Figure 3.8, the front curve of spreading G-Cell boundaries under the longest path computation would be maintained after the scaling. The placement will be compacted excessively for less congested regions. To resolve this problem, we alternatively assume $x_{i,n+1} = W \quad \forall i$, and make

Algorithm for solving the LP
Input: B-graph G(V,E)
Output: Maximized σ
begin
  $\sigma = 1$
  while(1)
    Perform Longest-path algorithm
    $L_p = MAX(x_{i,n+1})\ \forall i$
    if($L_p \leq W$)
      Break
    else
      $\sigma = \sigma \times \frac{W - L_h}{L_s}$
  end while
end

Figure 3.7 The longest path algorithm and iterative scaling for deciding boundary locations

$x_{i,1} \geq 0$ as the feasibility check. We could obtain two sets of x-coordinate result for each G-Cell boundary, let's say $x^l_{i,j}$ and $x^r_{i,j}$. The two sets of solution represent the two extreme cases in which the design is either packed to left or right. We thus name the two coordinates as valid range and the valid range will be used in determining the final boundary coordinates. Let $X_{i,j}$ denote the original G-Cell boundary coordinate ($X_{i,j} = (j-1) \times wTile$). If $X_{i,j}$ is within the valid range, it means the resulting packing is not too tight for both cases, and we will not move the boundaries ($x_{i,j} = X_{i,j}$). Otherwise, we move boundaries to the closer of $x^l_{i,j}$ or $x^r_{i,j}$.

$$x_{i,j} = \begin{cases} x^r_{i,j} & if \quad X_{i,j} > x^r_{i,j} \\ x^l_{i,j} & if \quad X_{i,j} < x^l_{i,j} \\ X_{i,j} & otherwise \end{cases} \tag{3.10}$$

Module Relocation

After adjusting each G-Cell boundary, the modules inside each G-Cell will be shifted accordingly. CROP updates the module location by maintaining the raio of distance to both boundaries before and after G-Cell resizing. As illustrated in Figure 3.9, $L1$ and $R1$ are the

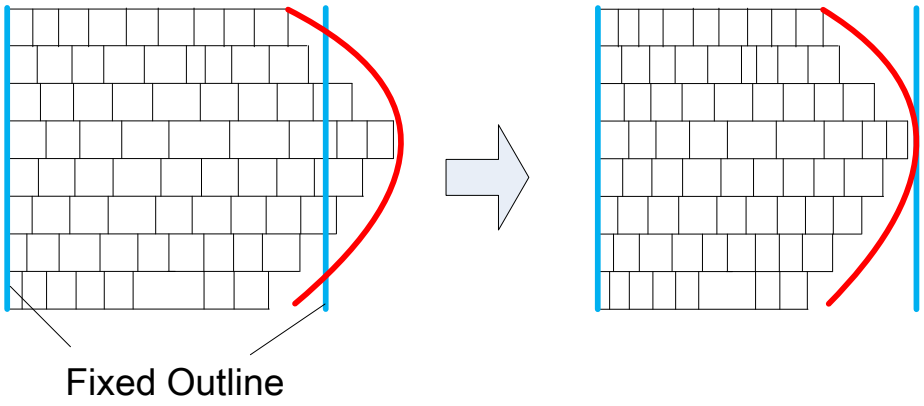

Figure 3.8 Problem of compacting design to left

original distances between the module to left boundary and right boundary respectively. Similarly $L2$ and $R2$ are the distances after G-Cell resizing. The module will be relocated to the place where condition $L1/R1 = L2/R2$ is met.

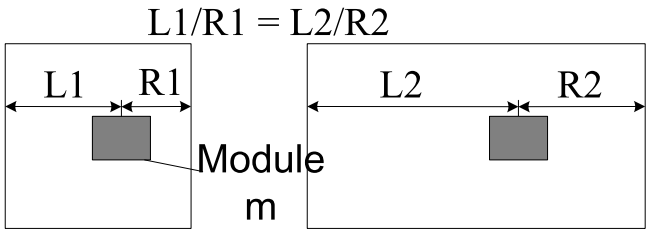

Figure 3.9 Module shifting illustration.

Macro Block Handling

In the traditional design flow, pre-placed blockages and IP-cores are first placed and their positions become fixed. Then the standard cells and movable macros are filled within the "gaps" between the blockages. Usually, certain amount of routing resources will be reserved for internal routing. The existence of big macros makes the routability issue more complicated. Hence we need to extend our method of handling standard cells to handling mixed-size placement solution.

Movable Macros In the case when the macros are movable, they can be legally relocated to the place of better routability. We apply a methodology similar to the one that we use to handle standard cells. But before we do that, we need be aware that macro blocks might cover multiple G-Cells, it is unlikely to shift them based on one G-Cell resizing. In other word, during the shifting, G-Cells covered by the macro may not be reshaped uniformly. Different G-Cell resizing solution might suggest totally different macro relocation. To tackle this challenge, we merge the covered G-Cells to become a super G-Cell. A macro will thus be repositioned based on the super G-Cell boundary coordinates. As in Figure 3.10, CROP merges the G-Cells that are covered or partially covered by the big macro.

After the merging, everything else goes the same way. The B-graph is generated to compute longest path. In the B-graph, since we merge G-Cells to become the super G-Cell, those merged cell boundary vertices are also merged or deleted. (The vertices along the super G-Cell boundary are merged, while the vertices inside the super G-Cell are removed) Similarly, the merged vertex is termed super vertex. The edge weights in B-graph are also necessarily updated. For instance, the area edge weight in between two super vertices are set to be the width of original super G-Cell width. The reason is macro cannot be squeezed, we need maintain a constant edge weight to accomodate the inside macros. With all the configurations mentioned, the computation of the longest path is the same as before except the two issues need be considered for the macro shifting.

1. *Reserved Routing Resource*

   Certain amount of routing resources above big macros need be reserved for internal routing purpose. This is also referred to as block porosity effect. To cope with block porosity, we increase the cost of the corresponding routability edges (E_r) in B-graph to compensate the capacity loss. In our experiment, we stick with the porosity reduction ratio used in generating ISPD07/08 global routing benchmarks.

2. *Non-Overlapping Guarantee*

   Although the overlapping between standard cells is permissible, big macros need be kept apart from each other. Otherwise, placement solution will be modified significantly if the

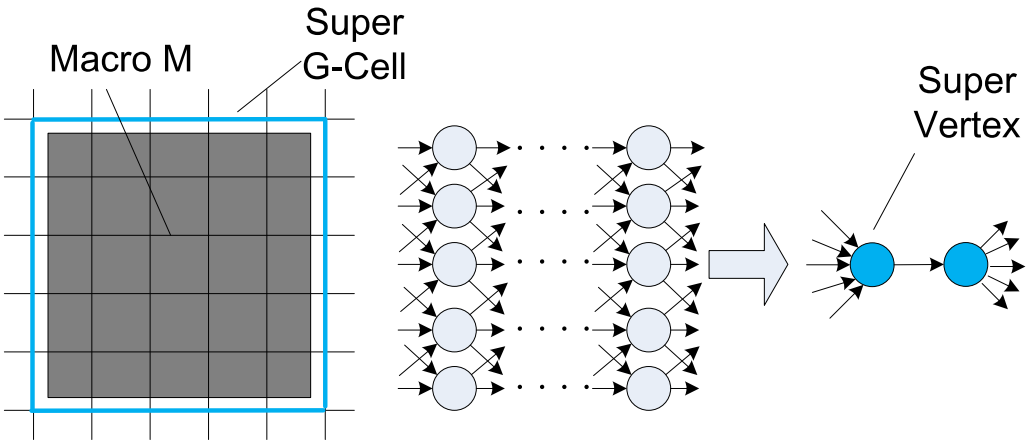

Figure 3.10 Merging of G-Cells for macro blocks.

legalization process is performed. Therefore, we place additional edges in the B-graph for guaranteeing the non-overlapping property between big macros. For instance, for adjacent macros, additional edges will be generated in between for making sure both parties do not overlap.

Fixed Macros Certain IP-cores and fixed macros are pre-placed on chip considering area, power, timing issues. In analog designs, there are design constraints that need be respected for macros such as mirroring constraint (e.g., two blocks are mutually viewed image for a fixed axis), alignment constraint, distance constraint, etc. It is therefore unallowed for moving the pre-placed big macros as they may degrade many design specifications and design constraints. In the above section, we have discussed how to manage movable blocks and how to embed the movable block consideration into current configuration. But in real designs, there are many fixed macros. We need incorporate fixed macro mode in order to better cope with complete mixed-sized placement.

We still incorporate the idea of merging (e.g., merge macro covered G-Cells into a super G-Cell). When there are fixed macros in the design, we need make sure the corresponding super G-Cell boundaries not moved during G-Cell resizing. Our methodology is embedded during construction of B-graph. We mark the super vertices as fixed by a boolean variable and record

its original position. These vertices should stay unmoved during shifting. When the algorithm proceed to a super vertex of fixed macro, and if it cannot be allocated without violating existing constraints. We detect the longest path to the macro and scale the path accordingly. Note that we keep the uniform scaling factor over the entire algorithm, which still seek the solution to the LP problem we originally propose. Figure 3.11 better illustrates the idea. Suppose the macro B cannot be assigned to the fixed location. And the dotted outline indicates the lowest position it can be placed. We first obtain the longest path leading to this macro and try to scale this path such that the macro covered vertices can be assigned to the fixed location. The detailed algorithm is shown in Figure 3.12. Please note that in some cases we could obtain very small σ due to fixed macros. In Figure 3.11, for instance, if the congestion between the two macros are very high (macro A and macro B), and the two macros are very close to each other, then the scaling of path for the fixed macro can potentially result in very small σ, which is the objective we try to maximize for the LP (Equations 1 - 6). In this case, the shifting may become ineffective because small value of σ will affect the magnitude of shifting and thus the degree of improvement. Then more optimization rounds and larger execution time is expected for obtaining desired results. We have conducted several experiments for fixed macro mode, the effectiveness of CROP depends on both the characteristics of the benchmark and the placer itself. those results will be shown in detail in the experiment section.

Congestion-Driven Detailed Placement

Detailed placement (DP) is commonly applied after global placement to improve HPWL for legalized placement solution. We develop a congestion-driven DP technique to compensate the netlength loss during the shifting stage and to further improve the routability. The flow of our proposed DP is shown in Figure 4.3 which contains congestion-driven global swap, vertical swap and local reordering. The whole DP flow is based on FastDP[43]. We only make the global swap step to be congestion-aware. The vertical swap and local reordering are local in nature and they contribute little to congestion reduction, so we still keep them HPWL targeted to retrieve netlength.

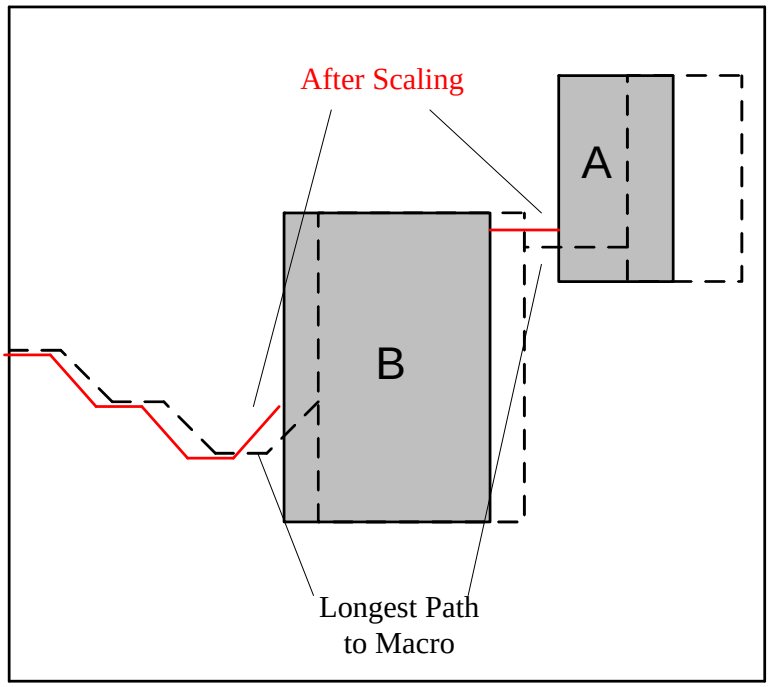

Figure 3.11 Path scaling for fixed macro case.

Congestion-Driven Global Swap for 2-pin Nets

Global swap step seeks to swap modules for improving HPWL based on a greedy pairwise position exchange. In [43], global swap is the step contributes most to the reduction of HPWL. As discussed earlier, HPWL could hardly reflect routability, especially when cells are swapped into highly congested regions. Although HPWL is reduced for this swap, but congestion might increase as the design becomes more dense. Therefore, we need consider the congestion during the swap activity. We thus change the swapping evaluation function to incorporate the congestion component. The HPWL is weighted by the congestion factor of α_n for net n. Simply put,

$$rHPWL_n = HPWL_n \times \alpha_n \tag{3.11}$$

In other word, the global swap is guided by the congestion weighted HPWL. In particular, if we swap standard cell A with standard cell B, the gain after swapping should be, $Gain_{A-B} =$

Algorithm for solving the LP with fixed macros
Input: B-graph G(V,E)
Output: Maximized σ
begin
  $\sigma = 1$
  while(1)
    Longest Path:
    Perform Longest-path algorithm
    foreach fixed super vertex v
    if $x_v \geq P_v$
      Detect longest path L_p^v leading to v
      $\sigma = \sigma \times \frac{P_v - L_h^v}{L_s^v}$
      goto: *Longest Path*
    if($L_p \leq W$)
      Break
    else
      $\sigma = \sigma \times \frac{W - L_h}{L_s}$
  end while
end

Figure 3.12 The longest path algorithm and iterative scaling for design with fixed macros

$\sum_{n\in N_A}(rHPWL_n - rHPWL'_n) + \sum_{n\in N_B}(rHPWL_n - rHPWL'_n)$. where N_A and N_B are the sets of nets that A and B are connecting to. $rHPWL'_n$ is the new cost after tentative swapping.

We will then discuss the way of getting the congestion weight. The straightforward approach, for instance, is to calculate the average congestion inside the bounding box. But this method is too rough to be reliable. So alternatively, we incorporate a more accurate model instead of simply lumping congestion together. We enumerate all possible Z routing paths inside the bounding box and calculate α_n by the average congestion along all the paths. Hence,

$$\alpha_n = \frac{w_{tol}}{E} = \frac{\sum_{p\in P}\sum_{e\in p} w(e)}{E} \tag{3.12}$$

In the equation, P is the set of all Z routing paths inside the bounding box. e represents each global routing edge along path p. E is the total number of global edges for all paths, and $w(e)$ is the weight of edge e. w_{tol} represents the sum of weight. $w(e)$ is calculated by Equation 13. If there is overflow, the weight will rise quadratically. If not congested, the weight is set to be constant to penalize wirelength.

$$w(e) = \begin{cases} 1 & if \quad O_e = 0 \\ (U_e/C_e)^2 & if \quad O_e > 0 \end{cases} \tag{3.13}$$

Speedup Technique based on Look-up Tables

The proposed method in Section IV-A would be very expensive in terms of runtime, since we need to add up the edge weights along all Z paths. For a $p \times q$ sized 2-pin net bounding box, the time complexity would be $O((p+q)^2)$. In order to speed up the computing of total weight, we propose a look-up table method. With look-up table, the timing complexity could be reduced to O(p+q).

Let $w_{tol} = w_h + w_v$, where w_h is the total weight of horizontal global edges and vice versa for w_v. Without loss of generality, let's discuss the calculation of w_h. Similar results can be derived for w_v. Figure 3.13 illustrates a 2-pin net in global routing grid graph. Let $e^h_{i,j}$ denotes each horizontal global routing edge in the global routing grid graph. We mark

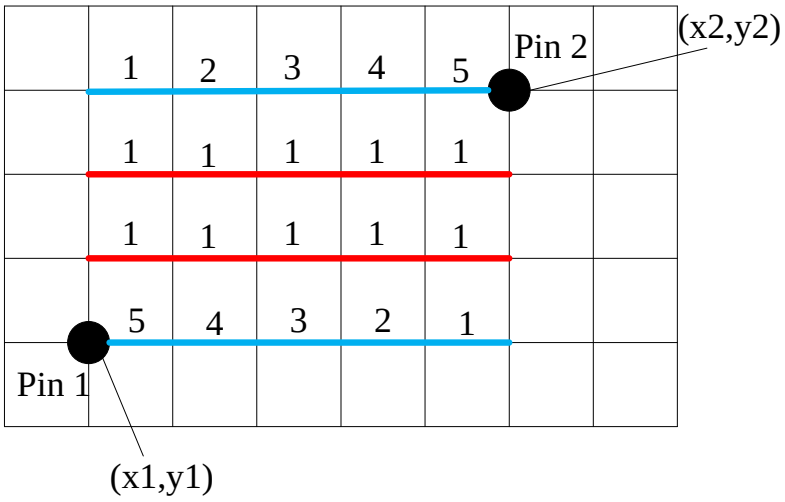

Figure 3.13 Number of Z routing paths passing through each horizontal global routing edge.

the number of Z routing paths passing through each horizontal global routing edge. Suppose the coordinates are (x1, y1) and (x2, y2) for pin 1 and pin2 respectively. As suggested by Figure 3.13, $w_h = w1 + w2 + w3$, where $w1 = \sum_{j=x1}^{x2-1}((x2-j) \times w(e^h_{y1,j})$, $w2 = \sum_{i=y1+1}^{y2-1} w(e^h_{i,j})$, and $w3 = (j - x1 + 1) \times w(e^h_{y2,j})$.

We introduce five $m \times n$ tables, $T1$, $T2$, $T3$, $T4$ and $T5$. Each entry in the table corresponds to one grid point in the routing grid graph. The meaning of entry (r, c) for each table is summeried in Table 3.1.

T1	$T1_{(r,c)} = \sum_{j=\{1,\ldots,c-1\}} w(e^h_{r,j})$
T2	$T2_{(r,c)} = \sum_{j=\{c,\ldots,n-1\}} w(e^h_{r,j})$
T3	$T3_{(r,c)} = \sum_{j=\{1,\ldots,c-1\}} (c-j) \times w(e^h_{r,j})$
T4	$T4_{(r,c)} = \sum_{j=\{c,\ldots,n-1\}} (j-c+1) \times w(e^h_{r,j})$
T5	$T5_{(r,c)} = \sum_{i=\{1,\ldots,c-1\},j=\{1,\ldots,r\}} w(e^h_{i,j})$

Table 3.1 Notation of look-up tables.

Based on the notations in Table 3.1, $w1 = T4_{(x2,y1)} - T4_{(x1,y1)} - (x2-x1) \times T2_{(x1,y1)}$, $w2 = T3_{(x1,y2)} - T3_{(x2,y2)} - (x2-x1) \times T1_{(x2,y2)}$, and $w3 = T5_{(x2,y2)} - T5_{(x1,y2)} - T5_{(x2,y1)} + T5_{(x1,y1)}$. With the help of five look-up tables, the computing of w_{tol} can be done very efficiently. Now the time complexity is $O(1)$.

All the proposed tables can be constructed very efficiently by dynamic programming. In Table 3.2, we also show how dynamic programming is performed. Basically, the computing of current entry can be broken into subproblems using the value of entry that has already been computed.

T1	$T1_{(r,c)} = T1_{(r,c-1)} + w(e^h_{r,c})$
T2	$T2_{(r,c)} = T2_{(r,c+1)} + w(e^h_{r,c})$
T3	$T3_{(r,c)} = T3_{(r,c-1)} + T1_{(r,c)}$
T4	$T4_{(r,c)} = T4_{(r,c+1)} + T2_{(r,c)}$
T5	$T5_{(r,c)} = T5_{(r,c-1)} + T5_{(r-1,c)} - T5_{(r-1,c-1)} + w(e^h_{r,c})$

Table 3.2 Apply dynamic programming to construct look-up tables.

Multi-pin Nets Handling

We need extend the 2-pin nets calculation to multi-pin nets. Without knowing the exact routing path and topology, it is impossible to weight the congestion for a multi-pin net with all potential routing soultions. Hence we derive the multi-pin net weighting model from a simplified method. Suppose the module has multi-pin connection and the multi-pin net without the module forms the bounding box B. If the module is inside B, the weighting is ignored. Otherwise a 2-pin net from the module to the nearest pin in B is introduced. The congestion weighting of the multi-pin net can thus be converted to a 2-pin net case.

Results and Discussion

All our experiments are performed on a machine with 2.4GHz AMD Opteron processor and 4G of memory. In order to better analyze the performance of CROP, we propose two sets of benchmarks: ISPD-GR and ISPD-DR. Both benchmark sets are derived from ISPD05/06 [21] [22] placement contest benchmarks. The ISPD-GR benchmarks evaluate the routability up to the global routing stage by the total number of overflow. The ISPD-DR benchmarks are detailed routing benchmarks and an industrial router is applied to report the congestion as well as design rule violations. In order to test the performance of CROP for the benchmark with fixed macro,

we fix all big macros in ISPD-GR and generate Fixed Macro (FM) version of ISPD-GR. There are totally four placers involved in the experiments. This comprehensive experimental data could thoroughly investigate the performance of CROP on routability improvement.

Effectiveness of Techniques in CROP

In this subsection we show the effectiveness of our techniques for routability improvement on two benchmarks of ISPD-GR: adaptec3 and bigblue1 generated by routability-driven NTUplace (R-NTUplace) [31] in Table 3.3. The purpose is to view the routability improvement during each execution step. The overflow is the global routing estimation result obtained from the technique mentioned in Section II-C. We have the following observations:

1. *Overall Congestion*

   The total overflow is consistently improved after each round of shifting.

2. *Congestion Overhead*

   The shifting in one direction will introduce congestion overhead for the other direction. For instance, when shifting modules in the X direction (to improve vertical overflow), the horizontal overflow becomes worse. The overhead comes from the extra mismatching horizontal wirelength. However, in general, the extra overhead is smaller than the gain, which contributes to the continuing improvement consistently.

3. *Legalization Issue*

   The placement solution during module shifting is not legalized. The congestion data cannot be fully trusted. Therefore we also show the overflow after congestion-driven global swap, at which step the design is legalized again. It shows that the congestion is improved over the original one, which better indicates the effectiveness of congestion reduction of proposed techniques in CROP.

4. *Congestion-Driven Global Swap*

   The global swap effectively reclaims lost wirelength during the shifting step. The shifting relocates the modules based on the tentative global routing solutions.(Section II-III).

The actually congestion after legalization is worsened. So congestion-driven global swap is actually an effective and necessary step, which reduce 15% to 25% wirelength in a short runtime.

	adaptec3				bigblue1			
	H o.f.	V o.f.	Total o.f.	Legal?	H o.f.	V o.f.	Total o.f.	Legal?
Before	53428	102804	156232	Y	50531	34409	84940	Y
Y shifting	40283	103637	143920	N	25320	45628	70948	N
X shifting	41168	93149	134317	N	31275	21525	52800	N
Y shifting	34239	96298	130537	N	23306	31703	55009	N
X shifting	37295	90350	127645	N	27025	19419	46444	N
Legalization	42548	93122	135670	Y	28223	20145	48478	Y
Global Swap	32335	81013	113348	Y	17238	17094	34322	Y

Table 3.3 Congestion reduction in CROP flow.

ISPD-GR Benchmarks

In this subsection we show the full experimental results on the ISPD-GR benchmarks derived from ISPD05/06 placement benchmarks. We utilize four different public available academic placers to generate the initial legalized placement solution. And we derive the corresponding global routing benchmark for the case with CROP and the case without CROP. To keep our comparison fair and complete, we follow the rules of ISPD07/08 [28] [29] global routing contest for determining the coefficients such as the block porosity effect etc.

In particular, the initial legalized placement solutions are generated by FastPlace3.1[39], NTUplace3[40], mPL6[41], and R-NTUplace[31]. In the experiment, FastRoute 4.0 [42] is utilized to report the global routing results. We have also tried NTHURouter[44] and the two routers produce similar results. We observe that FastRoute runs comparatively faster and the adoption could save experimental runtime.

Table 3.2 shows the results in detail. For each placer, we show the routing results before and after applying our tool. The entry with "/" means the original placement is routable so we do not apply CROP. More specifically, before applying CROP, there are 6, 11, 13, and

11 unroutable cases for the solutions of FastPlace3.1, NTUplace3, mPL6 and R-NTUplace respectively. After applying CROP, the number is reduced to 1, 1, 4 and 1. Out of these benchmarks, newblue3 is proved to be unroutable(more out pins than capacity in a G-Cell). So newblue3 cannot be reduced to overflow free. From these results, CROP is very effecitve in congestion reduction.

We also report the CROP execution runtime in Table 3.4. The runtime of our tool is trivial comparing with original placement runtime. For instance, the total runtime for mPL6 on our platform is more than 24 hours. While the total execution time of CROP is around one hour. Noticeably, the routing runtime is also saved considerably. The average speedup depends on different benchmark and specific placer. From our experiment, we could achieve roughly 6× speedup on average. The routing runtime improvement suggests the placement solution after applying CROP becomes easier to route.

Another aspect for evaluating CROP is the routed wirelength. After applying CROP, the total wirelength are 0.5% better, 1% better, 0.5%worse and 5% better for FP3.1, NTUplace3, mPL6 and R-NTUplace respectively. Generally speaking, the routed wirelength is on the same level with original design. But in many cases, we notice the routed wirelength becomes better. The reason is the new placement solution is easier to route, such that the router does not need to make huge detours and eventually saves the wirelength.

Fixed Macro Solutions

In order to better evaluate the performance of placement benchmarks with fixed macros, we create the FM (Fixed Macro) mode of the ISPD-GR benchmarks in which each macro is fixed. For the sake of runtime, we use global routing as routability checker. We conduct similar experiments as in Section V-B. In Table 3.4, we show the details of results on the FM mode of ISPD-GR benchmarks. For each test case, the row marked with "FM" is the corresponding benchmark in FM mode. As before, we list the overflow change with and without applying CROP, the CPU time of CROP, routing runtime and routed wirelength. First, we could observe that the congestion is consistantly improving. For each experimenting benchmark, the overflow

Metrics	Tools		a1	a2	a3	a4	a5	b1	b2	b3	n1	n2	n3	n4	n5	n6
Routing Overflow	FP3.1	w/o	/	1260	4	/	/	18755	769	/	/	/	9642	/	46	/
		w	/	0	0	/	/	0	0	/	/	/	9019	/	0	/
		FM	/	0	0	/	/	0	293	/	/	/	8998	/	0	/
	NTUPlace3	w/o	2885	2369	1621	141	/	15896	19793	15259	/	60	9442	3394	/	8
		w	0	0	0	0	/	0	0	0	/	0	8480	0	/	0
		FM	0	0	19	0	/	0	5928	10	/	0	8416	0	/	0
	mPL6	w/o	20	18535	22539	5703	7307	46995	1736	4678	/	9	8835	5649	12475	4495
		w	0	11289	660	0	0	0	0	0	/	0	8405	285	0	0
		FM	0	7465	12225	0	28	0	0	0	/	0	8220	0	10186	12
	R-NTUplace	w/o	51	2849	94	20	16	12887	38616	2264	/	898	10065	385	/	/
		w	0	0	0	0	0	0	0	0	/	0	8551	0	/	/
		FM	0	0	6	0	0	0	11670	0	/	0	8392	0	/	/
CROP CPU (second)	FP3.1	FM	/	92	200	/	/	73	193	/	/	/	297	/	406	/
			/	186	480	/	/	133	377	/	/	/	590	/	1056	/
	NTUplace3	FM	70	98	238	227	/	70	308	453	/	203	270	198	/	487
			169	196	605	446	/	122	416	977	/	456	455	366	/	945
	mPL6	FM	75	143	374	279	506	92	229	472	/	216	366	272	630	292
			142	213	491	559	967	165	457	887	/	554	774	466	1306	982
	R-NTUplace	FM	65	97	272	216	301	66	224	690	/	225	256	189	/	/
			121	168	456	435	765	111	394	1409	/	390	414	301	/	/
CROP iterations	FP3.1	FM	/	2	2	/	/	2	2	/	/	/	2	/	2	/
			/	4	4	/	/	4	4	/	/	/	4	/	4	/
	NTUplace3	FM	2	2	2	2	/	2	3	2	/	2	2	2	/	2
			4	4	4	4	/	4	4	5	/	4	4	5	/	4
	mPL6	FM	2	2	3	2	3	2	3	2	/	2	2	2	3	2
			4	4	5	4	5	4	4	4	/	5	4	4	5	4
	R-NTUplace	FM	2	2	2	2	2	2	2	3	/	2	2	2	/	/
			4	4	5	4	4	3	5	4	/	3	4	4	/	/
Routing CPU (second)	FP3.1	w/o	/	280	223	/	/	1492	1042	/	/	/	1331	/	154	/
		w	/	34	107	/	/	39	282	/	/	/	13832	/	32	/
		FM	/	33	215	/	/	36	762	/	/	/	13328	/	171	/
	NTUplace3	w/o	2031	311	1096	389	/	1945	2517	2692	/	362	12726	1271	/	1088
		w	144	20	101	26	/	46	87	80	/	28	13006	68	/	78
		FM	181	20	218	44	/	52	1298	445	/	33	13114	88	/	729
	mPL6	w/o	420	1478	6795	2859	2927	1941	448	2397	/	131	13661	2329	5756	6085
		w	58	974	1066	42	202	75	256	69	/	23	13554	1581	200	155
		FM	63	588	2919	120	732	129	148	94	/	28	13224	408	2697	964
	R-NTUplace	w/o	642	311	975	248	531	1619	3869	6439	/	933	12801	515	/	/
		w	101	17	108	24	47	41	303	136	/	31	13079	25	/	/
		FM	140	18	339	40	782	47	1713	1287	/	58	13009	29	/	/
Routed Wirelength ($\times 10e7$)	FP3.1	w/o	/	0.31	0.85	/	/	0.27	0.47	/	/	/	0.81	/	1.70	/
		w	/	0.31	0.85	/	/	0.26	0.48	/	/	/	0.84	/	1.65	/
		FM	/	0.31	0.89	/	/	0.26	0.47	/	/	/	0.83	/	1.73	/
	NTUplace3	w/o	0.30	0.30	0.84	0.71	/	0.28	0.47	0.83	/	0.46	0.74	0.81	/	0.95
		w	0.29	0.30	0.82	0.72	/	0.28	0.47	0.79	/	0.45	0.74	0.78	/	0.92
		FM	0.29	0.30	0.84	0.75	/	0.28	0.49	0.81	/	0.47	0.74	0.79	/	0.96
	mPL6	w/o	0.27	0.32	0.89	0.71	0.79	0.27	0.52	0.79	/	0.45	0.77	0.77	1.29	1.0
		w	0.26	0.32	0.85	0.72	0.77	0.26	0.54	0.77	/	0.45	0.77	0.78	1.40	0.96
		FM	0.26	0.32	0.91	0.75	0.80	0.28	0.52	0.79	/	0.46	0.76	0.77	1.34	1.07
	R-NTUplace	w/o	0.30	0.30	0.87	0.74	0.97	0.28	0.51	0.91	/	0.48	0.77	0.83	/	/
		w	0.29	0.30	0.83	0.73	0.86	0.27	0.49	0.84	/	0.46	0.76	0.79	/	/
		FM	0.29	0.30	0.87	0.74	0.96	0.27	0.52	0.92	/	0.48	0.75	0.80	/	/

Table 3.4 CROP results on ISPD-GR benchmarks and Fixed Macro(FM) mode

is better than the input placement solution. Second, although the congestion is improving, the enhancement is usually less compared with the case which allows movable macros. Since the range of movement is restricted when there are fixed macros, the required runtime (or refininig iterations) is longer. Third, the routed wirelength is not affected. After applying CROP, the total wirelength is 0.8% worse, 0.6% better, 0.8% worse and 2.0% better for FP3.1, NTUplace3, mPL6 and R-NTUplace respectively. Generally speaking, the original placement solution is well maintained.

ISPD-DR Benchmarks

ISPD-GR benchmarks evaluate the performance of CROP to the global routing stage. However, routability can be more accurately reflected in detailed routing stage. We derive the ISPD-DR benchmarks also from ISPD05/06 benchmarks in the LEF/DEF format. Due to the lack of publicly available detailed routers, we apply the industrial tool SOC Encounter for further verification. The placement solution with and without CROP are imported into Encounter and routed by built-in WROUTE. The detailed routing results are shown in Table 3.5.

We assign more restricted routing capacity by blocking out the whole capacity of macro from layer1 to layer4 (In ISPD-GR, we assign block porosity penalty to reduce the capacity of layer3 and layer4). The ISPD-DR benchmarks are not easy for WROUTE, which usually can not fix the routing overflow or violations cleanly. But after comparing the routing overflow with and without CROP, it is noticeable that routability is improved 10% to 20%. The improvement is not trivial as WROUTE is embeded with various features, and routability is only one of them. The total runtime, however, does not show positive feedback. The reason is no benchmark can be routed free of congestion, hence WROUTE has to spend considerate effort on ripup-and-reroute. The ripup-and-reroute procedure is controlled by a user-defined parameter for limiting the maximum round. The routed wirelength, in some cases, is improved. Similarly, it is because the routing becomes easier and unnecessary detours can be avoided.

Metrics	Tools		a1	a2	a3	a4	a5	n1	n4	n5	n6	b1	b2	total	norm
Routing Overflow	FP3.1	w/o	10644	8333	40493	51732	50486	10453	27918	88848	/	/	/	322384	1
		w	7433	6123	27100	38708	43332	8311	20655	62339	/	/	/	246107	0.7634
	NTUPlace3	w/o	9857	20515	46262	55901	57655	17876	58905	93534	81466	3593	66189	513518	1
		w	6804	14872	39396	44102	52998	11234	34570	88239	52678	2298	52617	400564	0.78
	R-NTUplace	w/o	9505	20128	47806	57897	59627	11499	36133	104753	84870	3583	71094	506895	1
		w	7575	16794	39023	53019	52324	8175	30987	92983	77654	2212	63220	443966	0.876
Routing CPU (second)	FP3.1	w/o	3319	2400	7939	7483	11729	2614	1203	3530	/	/	/	40217	1
		w	2394	931	7683	6423	12061	2606	1337	3073	/	/	/	36508	0.908
	NTUplace3	w/o	2553	881	7902	1716	11459	3387	1755	3604	3567	2689	3122	42635	1
		w	2679	896	6623	1208	9908	3219	1514	3360	2897	2809	2566	37679	0.884
	R-NTUplace	w/o	2590	866	8315	6854	13152	2414	1348	3767	3859	2771	3388	49324	1
		w	2617	895	8421	7047	10611	2421	1414	3954	3857	2612	3190	47039	0.954
Routed Wirelength (×10e6)	FP3.1	w/o	0.93	1.10	2.47	2.23	5.03	0.91	1.68	4.88	/	/	/	19.23	1
		w	0.94	1.05	2.53	2.31	4.99	0.92	1.67	4.86	/	/	/	19.27	1.002
	NTUplace3	w/o	0.96	1.04	2.46	2.18	4.92	0.94	1.62	4.89	4.12	1.24	1.89	26.26	1
		w	0.96	1.03	2.50	2.22	4.90	0.94	1.64	4.83	4.08	1.23	1.84	26.15	0.996
	R-NTUplace	w/o	0.97	1.03	2.54	2.26	4.78	0.88	1.67	4.89	4.13	1.15	1.90	26.2	1
		w	0.96	1.02	2.46	2.26	4.74	0.88	1.70	4.87	4.13	1.14	1.88	26.04	0.994

Table 3.5 CROP results on ISPD-DR benchmarks

Conclusion

In this work, we have presented CROP to improve routability for placement solution as a refinement process. Our tool is independent of any placer and router. The main techniques involves congestion-driven module shifting and congestion-driven detailed placement. We generate ISPD-GR and ISPD-DR benchmarks to test CROP's performance. And CROP shows promising results for various placement tools. We will further improve its performance and stability.

Acknowledgments

The authors would like to thank Dr. Yao-Wen Chang, the UCLA CAD group, and Dr. Igor Markov for the help with NTUplace3.0, mPL6 and Executable Placement Utilities respectively.

CHAPTER 4. RegularRoute: An Efficient Detailed Router with Regular Routing Patterns

A paper accepted by International Symposium on Physical Design

Yanheng Zhang and Chris Chu

Abstract

Detailed routing is an important phase of realizing exact routing paths for optimizing various design objectives and satisfying increasingly complicated design rules. In this paper, we propose RegularRoute, a fast detailed router trying to use regular routing patterns in a correct-by-construction strategy for better routability and design rule satisfaction. Given a 2-D global routing solution and the underlying routing tracks, we generate a detailed routing solution in a bottom-up layer-by-layer manner. For each layer, the routing tracks are partitioned into a number of panels. We formulate the problem of assigning global segments into different tracks of each panel as a Maximum Weighted Independent Set (MWIS) problem. We propose a fast and near-optimal heuristic to solve the MWIS problem. Then unassigned segments after MWIS are partially routed by a greedy technique. For the unrouted portion of each segment, its terminals are promoted so that the assignment is deferred to upper layers. At top layers, we apply panel merging and maze routing techniques to achieve better routability. Due to the unavailability of academic detailed routing benchmarks, we proposed two sets of detailed routing testcases derived from ISPD98[45] and ISPD05/06 [21, 22] placement benchmark suites respectively. The experimental results demonstrate that RegularRoute significantly outperforms WROUTE in both quality and runtime.

Introduction

Because of the problem complexity, VLSI routing is usually divided into global routing and detailed routing. In global routing stage, rough routing decision is made based on G-Cell-to-G-Cell (e.g., global routing cell) connection on a global routing grid graph. Detailed Routing, on the other hand, realizes exact routing paths considering geometrical constraints based on the global routing solution. Detailed routing is an important stage in the sense that it is directly related to the routing completion and design rule satisfaction. It also impacts many design metrics such as timing, manufacturability, etc.

Detailed routing has been well studied since 70's (e.g., [13, 14]) but the topic is not frequently seen in recent publications. For modern designs in which over-the-cell routing is applied, the most common technique for detailed routing is rip-up and reroute such as the one in Mighty [46]. However, such a simple sequential approach is not sufficient to handle congested designs and it usually creates unnecessary detour. DUNE [15] and MR [16] proposed to handle full chip gridless routing by similar multilevel approaches, in which the routing undergoes a coarsening phase and an uncoarsening phase. But these multilevel routers still rely on sequential rip-up and reroute technique and nets at the upper levels of hierarchy are routed based on inaccurate information. Besides, how to do layer assignment is not fully discussed. Nam et al.[17] proposed a detailed router for FPGA based on boolean satisfiability. Though the approach achieves good solution quality, the runtime is extremely long. Zhou et al. [18] introduced track routing as an intermediate step between global routing and detailed routing. In track routing, segments extracted from global routing solution are assigned to routing tracks. But track routing does not consider the connection of a segment to pins or segments in other layers. These issues are postponed to detailed routing, which may fails to connect different parts of a net. Mustafa [19] presented a meaningful technique to perform escape routing for dense pin clusters, which is a bottleneck of detailed routing. A multi-commodity flow based optimal solution and a Lagrangian relaxation based heuristic are proposed. Nevertheless, the technique is not proposed for solving the whole-chip detailed routing.

With diminishing feature size, many complex design rules are imposed to ensure manufac-

turability. It has been reported that for 32nm process, the number of rules reaches several thousands [1] and the design rule manual has roughly a thousand pages [2]. The dramatic increase in the number and complexity of the design rules makes detailed routing progressively complicated and time-consuming. We notice that many of those complex rules are triggered only by non-trivial routing patterns. Here we define regular patterns as those avoiding jogs and unnecessary detours as much as possible. Figure 4.1 illustrates two routing solutions for the same problem. The top one is irregular routing with many jogs and detours while the bottom one is regular routing which only uses simple patterns. If only regular patterns are used, it is not even necessary to check many design rules and the routing solution will be correct by construction. On the other hand, if a routing pattern is irregular, even though it may not violate any design rule, it is likely to be detrimental for both yield and routability. Moreover, regular routing introduces less vias, jogs and wirelength and hence is better in terms of timing, signal integrity and power consumption. In order to reduce the implementation complexity and runtime of detailed routers and to improve the electrical properties, yield and routability of circuits, we propose to perform detailed routing based on regular patterns. Note that this approach is along the lines of the restrictive design rule approach that the industry has started applying to the device layers to enhance manufacturability. In this paper, we extend it to the interconnect layers.

Potentially, regular routing may adversely affect routability because it is more restrictive and may be less effective in resolving congestion. This paper shows that the opposite is true. The reason is that for regular routing, with an appropriate algorithm, the solution space can be explored much more effectively and efficiently. On the contrary, for routing with general patterns, the solution space is much larger. But it is also much harder to be explored. The best known approach is to route the nets one by one using maze routing together with rip-up and reroute. Such an approach is very time-consuming (especially if complicated design rules need to be checked repeatedly throughout the routing process) and is prone to getting stuck in local minima.

In this works, we present a fast and effective algorithm called RegularRoute to perform

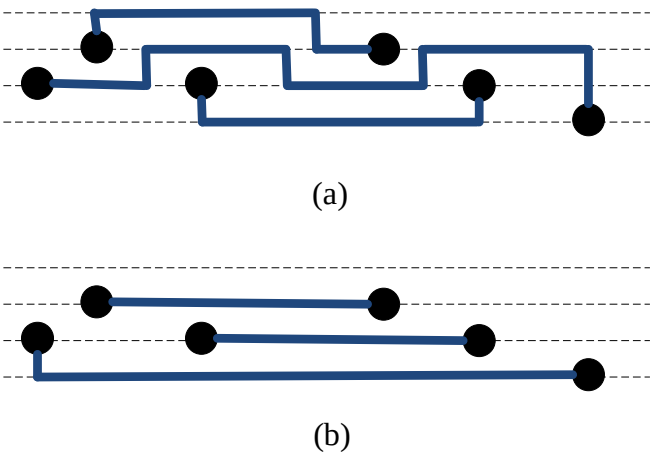

Figure 4.1 (a) Non-trivial routing patterns. (b) Regular routing patterns.

detailed routing with regular patterns. Novel techniques in RegularRoute are listed below:

- We introduce a new bottom-up layer-by-layer framework for detailed routing.
- We propose a single trunk V-Tree topology for routing local nets.
- We divide the routing problem of global nets into assignment of global segments into panels. This approach facilitates parallel processing as assignment for different panels are independent of one another.
- We formulate the global segment assignment problem for each panel as a Maximum Weighted Independent Set (MWIS) problem. This formulation enables all segments to be considered simultaneously.
- We present a fast and near-optimal heuristic to solve MWIS.
- We employ a technique to maximize the usage of a panel by partially assigning some of the remaining segments after MWIS.
- We introduce a terminal promotion technique to connect various segments of each net assigned to different layers.

- We present panel merging and maze routing techniques to handle unassigned segments at the top layers.

We implemented RegularRoute and tested its performance on two sets of detailed routing testcases derived from ISPD98[45] and ISPD05/06 [21, 22] placement benchmarks respectively. Experiments show that RegularRoute significantly outperforms WROUTE in both quality and runtime. In particular, RegularRoute completes routing of all eight ISPD98 derived testcases and achieves much better routability in ISPD05/06 derived testcases.

The rest of paper is organized as follows: Section 2 provides the problem formulation and an overview of RegularRoute. Section 3 discusses the routing for local nets. In Section 4, we introduce techniques for handling global segments assignment. Experimental results are shown in Section 5.

Materials and Methods

Preliminaries

In this section, we will present some definitions, the problem formulation and the algorithm flow of RegularRoute.

Problem Formulation

In this paper, as regular routing is considered, we model the routing resource as a 3-D regular grid graph. Each grid edge can accommodate one wire except for edges with blockage, which cannot be used. Each layer of the graph has a *preferred routing direction* and the preferred directions of adjacent layers are perpendicular to each other. We assume the preferred direction of lowest layer (metal1) is horizontal. For each layer, the routing usage that is in the preferred direction is called *preferred usage*. Otherwise, the routing usage that is perpendicular to the preferred direction is called *non-preferred usage*. A sequence of unblocked grid edges along the preferred routing direction of each layer is called a *routing track*.

Assume a placed netlist with exact pin locations and a corresponding 2-D global routing solution are given. In this paper, we assume all pins are on metal1. The detailed routing

problem is to route all nets on the grid according to the global routing solution such that routes of different nets do not intersect. The primary objective of detailed routing is to complete as many nets as possible. The secondary objectives include minimizing non-preferred usage, via count and wirelength. In industrial applications, there may be many other design metrics such as timing, crosstalk, yield, etc. These metrics can potentially be incorporated into our framework but they will not be handled directly in this paper.

In our framework, the global routing solution of each net is partitioned into a set of *segments* by breaking it at the turning points. Each segment is a horizontal (or vertical) route which spans multiple G-Cells in a row (or column). Then detailed routing of global nets is formulated as assigning the global segments to the routing tracks. Ideally, each segment should be assigned to one track. In order to make routing less restrictive, assigning a segment to more than one tracks connected by short non-preferred usage or via is allowed but discouraged. We define a *panel* to be the collection of all tracks on one layer within one row (for odd layer) or one column (for even layer) of G-Cells. Note that each segment can only be assigned to tracks on a deck of panels that are on different layers but are associated with the same row/column of G-Cells spanned by the segment. In other words, it is natural to perform the global segment assignment in a panel-by-panel manner. Figure 5.1 shows the definitions of track, segment and panel.

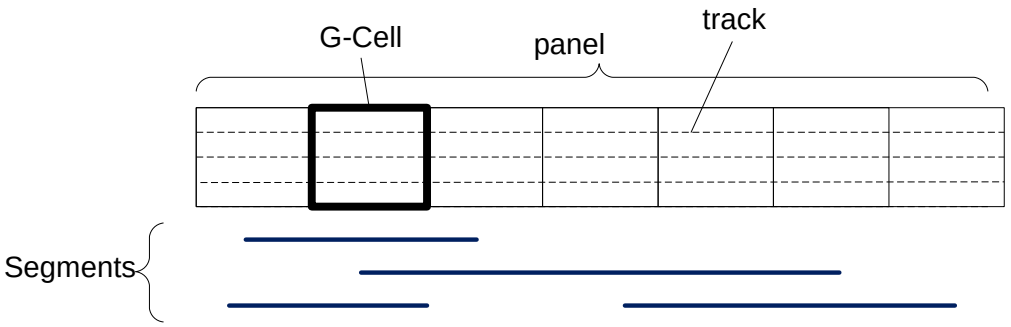

Figure 4.2 Definitions of track, segment and panel.

Algorithm Flow

We show the flow of RegularRoute in Figure 4.3. RegularRoute starts with extracting global segments from the 2-D global routing solution. Then local nets are pre-routed using the single trunk V-Tree topology. In the following global segment assignment, the routes of local nets are treated as blockages. Next, we perform global segment assignment in a bottom-up layer-by-layer manner. At each layer, the segment assignment of different panels are handled independently. For each panel, we formulate global segment assignment as a MWIS problem and solve it by a heuristic with two rounds. In the first round, we restrict the number of choices for each segment to estimate the level of difficulty of the panel. If it is hard to assign all segments, we proceed to the second round with the same choices for the assigned segments but full choices for the unassigned segments in the first round. Then the MWIS problem is solved again. After that, we apply a partial assignment technique to increase the utilization of the panel. Then if we have not reached the top horizontal or vertical layers, for the unassigned segments, we promote their terminals and defer their assignment to upper layers. For the unassigned segments at the top layers, we utilize a panel merging technique which provides more flexibility in the assignment by allowing segments to derivate from the global routing solution. Finally, maze routing is applied for the residual unassigned segments.

Local Net Routing

In detailed routing, the net or sub-net that resides totally inside one G-Cell is called a local net. In RegularRoute, local nets are routed before assigning global segments. The routing solutions of the local nets are treated as blockages in the following global segment assignment. It is possible to route local nets and global segments simultaneously by integrating local net routing into the MWIS framework in Sec. 4. But in order to reduce the size of MWIS problems and hence the runtime of the whole algorithm, we choose to handle local nets before global segments.

In this section, we first introduce local net routing by single trunk V-Tree topology. We then demonstrate that single trunk V-Tree topology can better preserve routing resources to be used

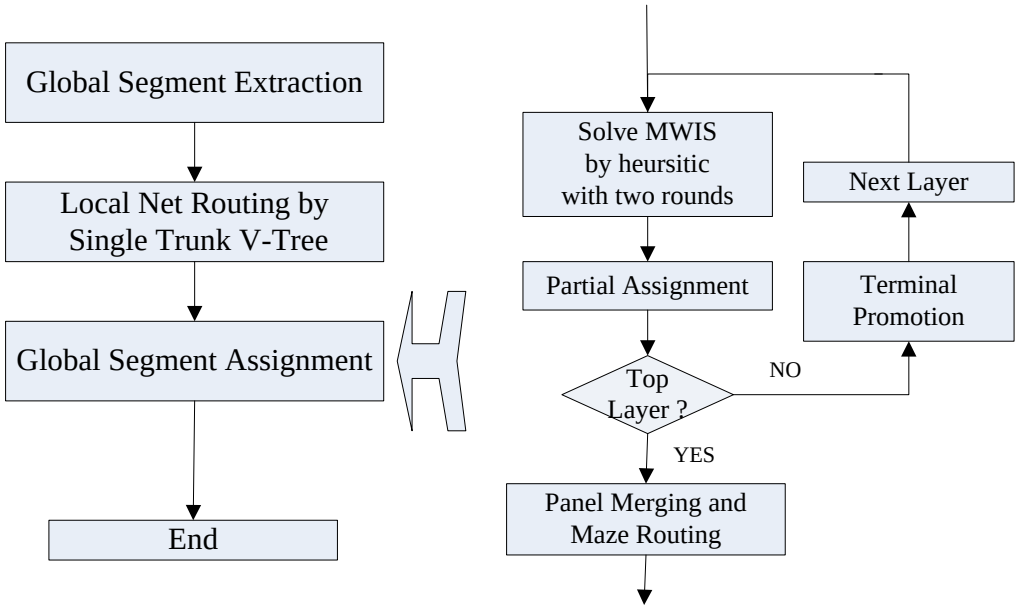

Figure 4.3 Flow chart for RegularRoute.

in global segment assignment. It is possible to have conflicts among the single trunk V-Trees of different nets. Hence we also present a branch and trunk shifting technique to resolve the conflicts.

Single Trunk V-Tree Topology

Single trunk tree has been proposed to predict the routing usage or interconnect properties at early design stages [47]. In here, we use a single trunk tree with the trunk being vertical to route the local nets. We call this topology *single trunk V-tree*. Consider all the pins inside a G-Cell. The x-coordinate of the trunk is set to be the median of the x-coordinates of all pins. The trunk spans from the minimum y-coordinate to the maximum y-coordinate of all pins. The trunk is on metal2. We connect each pin to the trunk with a metal1 (horizontal) connection, which we call a branch, and a via. Figure 4.4 shows an example of single trunk V-Tree.

Single trunk V-Tree is very easy to construct. The time complexity is $O(N \log N)$ for a local net with N pins. In our testcases, the average pin count is very small (around 3). So the runtime is negligible compared other steps.

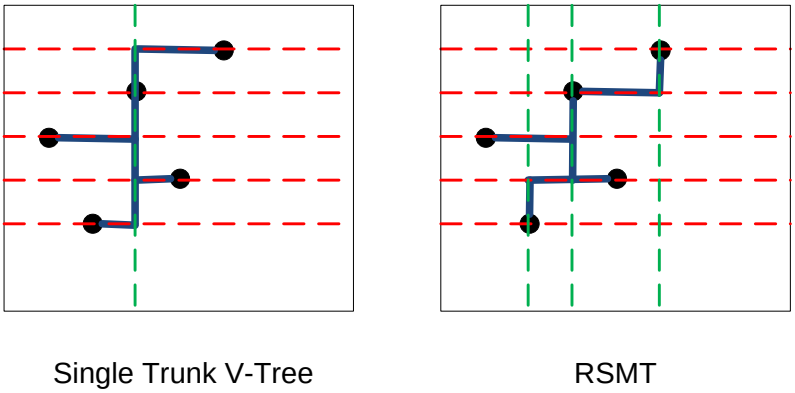

Figure 4.4 Track blockage count for single trunk V-Tree and RSMT.

There are many candidate topologies to construct the trees for local nets. For instance, RSMT and RMST are promising candidates. The reason we choose single trunk V-Tree is for the sake of saving routing resources on metal2. In Figure 4.4, single trunk V-Tree and RSMT are presented for the same 5-pin net. In metal1, five tracks are blocked in both cases. In metal2, for single trunk V-Tree, only one metal2 track is blocked but for RSMT, three metal2 tracks are blocked. As single trunk V-Tree blocks fewer tracks on metal2, global segment assignment will have more tracks to use later on.

Trunk and Branch Shifting

During the local net routing, we first determine the vertical trunk. If the total pin number is odd, the trunk has only one choice for minimum wirelength. Each branch has only one choice too. If there are multiple local nets inside one G-Cell, there is a risk of conflict among trees of different local nets. To avoid the conflict, we apply trunk and branch shifting by trying neighboring tracks. Any unresolved conflict can be resorted to higher layers (e.g., metal3 and metal4). But in our experiments, all local nets can be routed using only metal1 and metal2. The results will be shown in Section 5.

Global Segment Assignment

In Section 2, we have discussed the flow and general formulation of global segment assignment. In this section, we cover the details. We first present the detailed MWIS formulation for assigning the global segments to a panel using regular routing patterns. We then provide a fast and effective heuristic to solve the MWIS problem. We next discuss the partial assignment to increase the utilization of the panel. Then we talk about the terminal promotion techniques to defer the unassigned segments to upper layers. For the unassigned segments on the top layers, we develop the effective panel merging and maze routing techniques for final improvement.

Segment Assignment by Maximum Weighted Independent Set

In Section 2.1, we have discussed the general problem formulation. And we have mentioned the global segment assignment problem in each layer is solved by a panel-by-panel strategy. The segment assignment inside one panel is a fundamental component to the whole algorithm. In this subsection, we will investigate this problem. Without loss of generality, we only consider horizontal panels (metal1, metal3, etc.).

We need present more definitions for global segment here as it is the major subject we handle in this section. The global segment is a route spans a row or column of G-Cells. We call starting G-Cell and destinating G-Cell the two ending G-Cells (or two ends) of the segment. For each end, it could be incident to a terminal or a pending segment. Being incident to a terminal means the segment needs be connected to the terminal inside the G-Cell. The terminal is either a metal polygon or a pin. Connecting the assigned segment and its terminal is called *terminal connection*. On the other hand, being incident to a pending segment suggests the end of the segment needs be connected to another segment which has not being assigned yet. The terminal, pending segment and terminal connection concepts are illustrated in Figure 4.5. In the figure, the left and right ends of segment 1 are incident to a existing metal polygon and a pin respectively. We show the terminal connection of the right end. The left end of segment 2 is incident to a pending segment that has not been assigned. The pending segment is shown in dotted line.

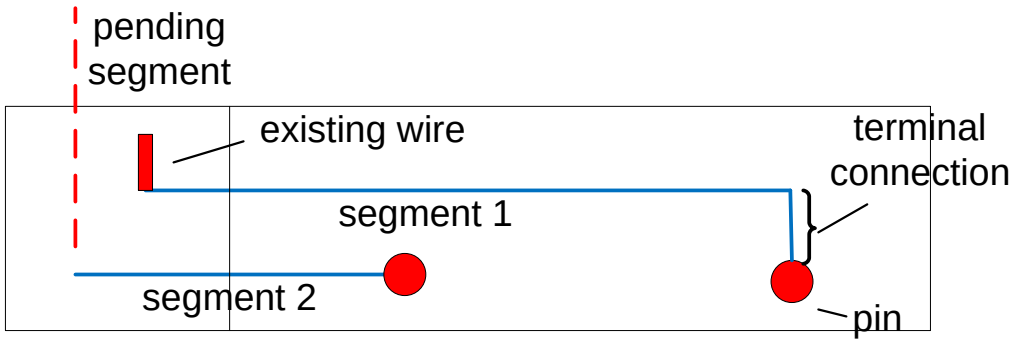

Figure 4.5 Concepts for incident terminal, pending segment and terminal connection.

To assist the formulation of the MWIS problem, we introduce the concept of choice when assigning a segment. A *choice* is a valid candidate solution to assign the segment using a regular routing pattern when no other global segments is considered. The candidate solution includes which track is used and how terminal connection is performed. In particular, we introduce a triplet (t, T_1, T_2) to represent a choice for a segment. t is the track used to assign the segment. T_1 and T_2 are the labels for both ends containing the information on how to perform terminal connection. Take T_1 for instance. If the first end is not incident to any terminal, T_1 is labeled as "none (F)". If it is incident to terminal and terminal connection will be performed in current layer, T_1 is labeled as "current (C)"; Otherwise, if terminal connection is not performed in current layer, T_1 is labeled as "next (N)". We also define the numerical values associated with the content of a label: "none (F)" and "next (N)" is **0** and "current (C)" is **1**. Basically, the numerical values are introduced to calculate the weight of a choice. The weight calculation will be introduced later.

In Figure 4.6, we show two segments with one choice and two choices respectively. In the triplets, the track and label for both ends are displayed. For segment i (the right one), choice c_i1 uses track t1 and does not connect the first terminal in current layer (labeled as "N"). And choice c_i2 uses track t3 and both terminals are connected in current layer (labeled as "C"). If there are conflicts between different choices, they cannot be selected together. For instance, in Figure 4.6, choice c_j1 conflicts with choice c_i2. Plus, the choices that derive from the same

segment mutually conflict with each other. For example, choice c_i1 conflicts with choice c_i2.

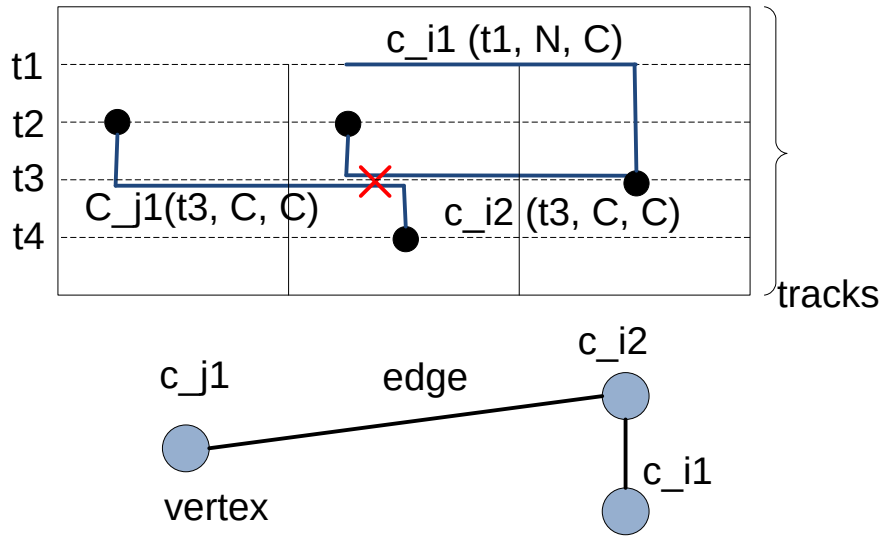

Figure 4.6 Conflicting choices and conflict graph.

We formulate a MWIS problem for solving the global segment assignment inside one panel. For each choice, we create a vertex. And the conflicts between choices are modeled as edges in the graph. One example of the conflict graph is shown in Figure 4.6. Each vertex is assigned a weight specifying the priority of the choice. The objective of the MWIS is to select a set of independent vertices with maximum weight.

The weight calculation for each vertex is important in the MWIS problem. It should contain components for both the *routing completion* (primary objective) as well as the *routing quality* for minimizing non-prefered usage, via count and wirelength (secondary objectives). We use the following function for weighting a vertex (choice).

$$W(v) = L + \alpha_1 \times (T_1 + T_2) + \alpha_2 \times (\frac{\sum_{b \in B} (D_b)^2}{\|B\|}) - \alpha_3 \times d + \alpha_4 \times (f_1 + f_2) \tag{4.1}$$

We have five major components for determining the weight of a choice.

1. *Segment Length*

   L is the number of global routing grids that the segment spans. It relfects length of the

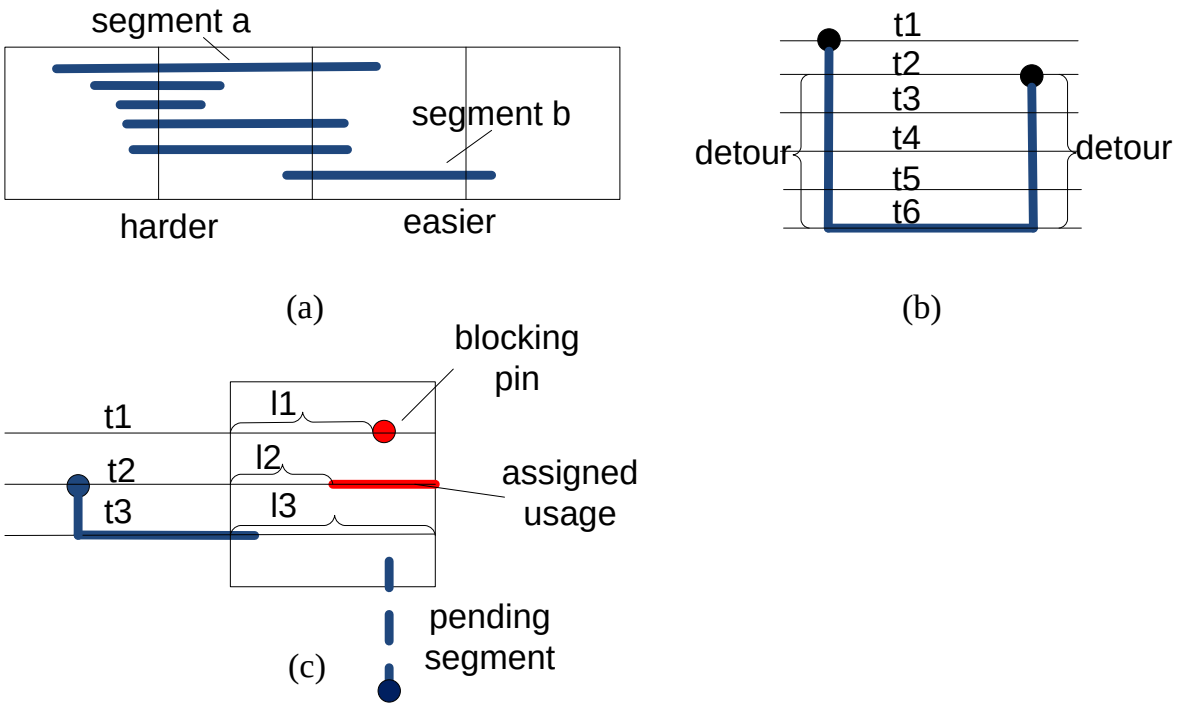

Figure 4.7 (a) G-Cell boundary density. (b) Detour component. (c) Flexibility component

segment. In the weight calculation, we encourage packing more usage to current layer. This is a component for routing completion.

2. *Terminal Connection*

This component increases the weight for the choices that perform terminal connection in current layer (label with "C"). Performing terminal connection in current layer preserves routing resources in the upper layers and saves additional vias. Though a necessary component, it should be noted that terminal connection is not always preferred. For instance, in Figure 4.7(b), the choice indicates connecting both the incident terminals in current layer. Based on terminal connection component, we will increase the weight for this choice. However, it should be discouraged as it introduces long non-preferred usage which might block other segments. In order to achieve the balance, we adjust this component by the detour component that will be introduced later. Terminal connection works for both routing completion and routing quality.

3. *G-Cell boundary density*

   This component is used to increment weight for the segments that cross dense G-Cell boundaries. We use the number of crossing segments to represent the boundary density. Intuitively, the segment passing through denser G-Cell boundaries is harder to assign. They can easily incur conflicts with other segments. We use the average quadratic G-Cell boundary density for this component. In Figure 4.7(a), segment a is harder to assign than segment b as it passes through denser G-Cell boundaries. In Equation 1, B is the set of G-Cell boundaries the segment passes through. D_b is the density of boundary b. We sum the quadratic value of density and divide it by the number of boundaries. This component is proposed for routing completion.

4. *Detour Component*

   We use d in the Equation1 to represent the detoured wirelength. The detour is defined as the extra wirelength over the minimum necessary wirelength to complete the assignment. For each segment, we define the tracks without causing any detour as the *preferred tracks*. Please refer to Figure 4.7(b). The segment is assigned to track $t6$, which creates a detour of 8 units. Long terminal connections will be made either in current layer or the other layers, which creates more blockages for future assignment. Hence the choice with long detour should be discouraged. The detour component is for routing quality.

5. *Flexibility Component*

   The flexibility component is used to differentiate choices for the segment with one or more ends that are incident to pending segments. As is discussed earlier, the pending segment has not being assigned. In this case, the track which offers more flexibility for assigning the pending segment is better of routability. We define the length from the close boundary of the ending G-Cell (left boundary for right end and right boundary for the left end) to the first blockage or the far boundary for a particular track as the flexibility length. In Equation 1, we use f_1 and f_2 to represent the flexibility lengths of both ends if certain track is taken (flexibility length will be 0 if the end is not incident to pending segment). As in Figure 4.7(c), $l1$, $l2$ and $l3$ are flexibility length of the right end for track

$t1$, $t2$ and $t3$ respectively. The track with larger flexibility length is more flexible (i.e., more opportunities) for assigning the pending segment. As illustrated in the figure, if we assign the segment to track $t1$ and $t2$, we cannot assign the pending segment as shown in dotted line without detour. The flexibility component is for routing completion.

In the equation, there are four coefficients (α_1, α_2, α_3 and α_4) for tuning the importance of each component. They are determined by experiment.

The MWIS problem is NP-Complete [3], and solving it optimally is time-consuming (there are hundreds of panels for each layer and each panel contains thousands of segments). Instead we develop an efficient heuristic. For each panel, we rank the vertices according to the cost function as shown in Equation 2,

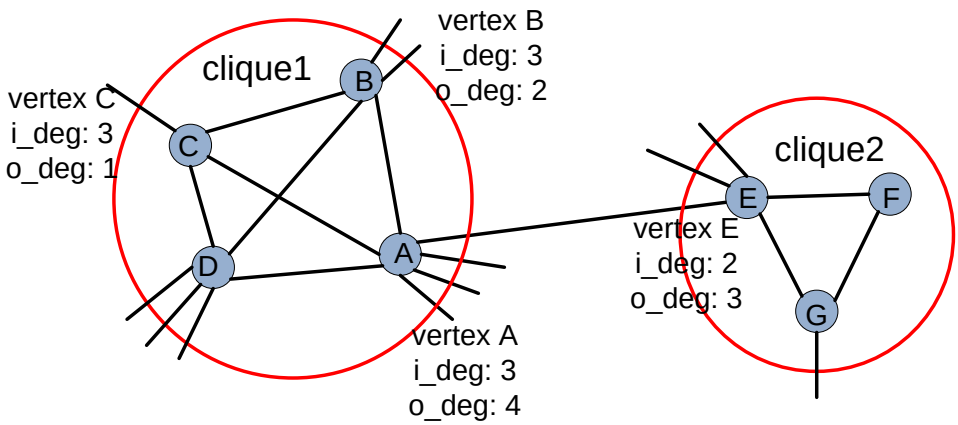

Figure 4.8 Calculation of inside/outside degree for each vertex.

$$C(v) = W(v) - \beta \times i\_deg(v) - \gamma \times o\_deg(v) \tag{4.2}$$

where W represents the weight of vertex v, i_deg is called inside degree and $o\_deg(v)$ is outside degree. The two degrees are defined based on *clique*. The clique is the set of vertices that derive from the same segment. For the particular vertex, the number of connection inside the clique is the inside degree and vice versa for outside degree. The definition is illustrated in Figure 4.8. Vertex A, B, C and D form $Clique1$ and we show the inside degree and outside degree for A, B and C respectively. The other three vertices form $Clique2$ and similar info is illustrated.

Basically, vertex with larger weight and smaller inside/outside degree is prioritized. First, the vertex with higher weight is preferred since our algorithm tries to pack as much usage as possible in current layer and the weight function (Equation 1) encourages this objective. Second, larger inside degree means more choices for the same segment. We could delay its assignment and leave the routing resources for less flexible ones. Third, the vertex with larger outside degree means more conflicts with other segments. Such choice is suppressed to allow less conflicting segments to be assigned earlier. Again, the coefficients (β and γ) is experimentally determined and we use a uniform set of coefficients for all testcases.

We assign the vertex with the order of decreasing cost. We assign the segment based on the choice suggested by the vertex. After processing each vertex, we perform graph update to deactivate the incident vertices. Basically all neighboring vertices that are connected to the processing vertex will be marked as inactive (dead vertices) and they will not be considered in future assignment. We will also update the inside/outside degree for the vertices that are incident to the neighboring vertices (except the processing vertex). And those vertices will be re-ranked using binary search. The assignment process goes on until all vertices in the graph are visited.

In our tool, the number of choices for each segment has big impact on runtime as it decides the number of vertices of the conflict graph. Providing small number of choices would lower solution quality while granting large number of choices might cost long runtime. To balance this effect, we implement the MWIS problem in two-round flow. The first round is an initial estimation of difficulty of the panel. In that regard we only consider the choices that are close to the preferred tracks. For instance, we set a bound of offset number of tracks away from the preferred tracks (We use 2 in the experiment). If in the first round all global segments in the panel can be assigned without any conflict and design rule violations, we dump the solution and move forward to the next panel. Otherwise, we grant unassigned segments with full choices. And the assigned segments will be ripped up and still use the same number of choices as before. Then the MWIS problem is solved again. This idea is a trade-off between solution quality and runtime. The algorithm might become prohibitively slow when the number of choices are too

large. In real practice, we could extended the idea to multi-round. The unassigned segments are granted with more and more choices gradually during the multi-round estimation. The number of tracks for each panel, however, is relatively small (around 10). A two-round strategy is sufficient for runtime speedup.

Partial Assignment

After solving the MWIS problem by our algorithm, there are potentially large number of remaining unassigned segments in the congested panels. In order to better utilize the routing resources of current layer, we need explore more possibility beyond the choices defined in the MWIS problem. We implement a partial assignment for increasing the utilization of the panel.

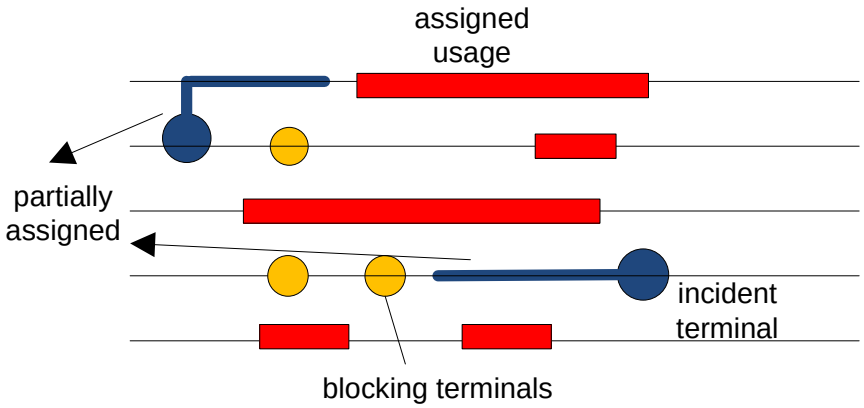

Figure 4.9 Partial assignment technique for unassigned segments.

For each unassigned segment with at least one end with incident terminal, we will try partial assignment starting from the incident terminal to the other end near the preferred tracks for optimizing the weighting function used in the MWIS problem (Equation 1). The idea is illustrated in Figure 4.9. Both ends of the unassigned segment are incident to terminals. The incident terminals are highlighted with bigger shape than the "blocking terminals" as in the figure. The previously assigned usage is treated as "blockage". We denote the partially assigned segments as well.

There is a special case When both ends are incident to terminals. It is possible the partial assignment originated from each end extends beyond each other. In this case, we will try assigning a non-preferred connection to link the two partial assignments. If the non-preferred usage can be assigned without blockage, the segment is assigned in current layer. Otherwise, we promote the far end of the partial assignment to upper layer as a new terminal. The terminal promotion will be introduced next subsection.

The partial assignment method offers more flexibility and is useful for further improving the utilization of current layer. An alternative idea is to incorporate the partial assignment into the MWIS problem. It is possible to make various partial assignment as choices in the conflict graph. However, it requires incorporating much more vertices and edges. We apply the partial assignment as postprocessing after MWIS problem.

Terminal Promotion

For the unassigned segments after applying partial assignment, we need defer their assignment to upper layers. However, there will be terminal connection issue when the segment is assigned in upper layer while its terminals are located in lower layers. Let's suppose a horizontal segment is finally assigned to metal5 and one of its end is incident to terminal in metal1. In this case, we have to connect the terminal to the assigned segment, with a set of short wires and vias between metal1 and metal5. However, if the routing resource is limited (i.e., nearby routing tracks are taken up), realizing this connection can be headache and may finally results in big effort in rip-up and reroute. In order to avoid this situation, we can promote the terminals of unassigned segments before we move to upper layer. After promoting terminals up, in the assignment in upper layer, we could always treat the terminals as in current layer. The idea could be highly effective for congested panels where the routing resources are limited. Hence the main difference of our algorithm and the track assignment[18] is that ours could guarantee a valid solution after all segments are successfully assigned. Yet track routing may spend a lot of rip-up and reroute effort for correcting failed terminal connection and segments between different layers.

We count the number of tracks that the terminal can access in the upper layer. If there is only one track that the terminal can access. We select the track and promote the terminal accordingly. If the terminal is capable of accessing multiple tracks, we pick the track that is closest to the terminal. If there is no access to upper layer, we rip-up the usage that is small and close to the terminal to guarantee it has at least one access. We first use a short connection on the track where the original terminal is located to where a via can be inserted. Then the via is used to promote the old terminal to the upper layer tracks. In the upper layer, we create a virtual stripe covering all the tracks in the upper layer that the original terminal can access (virtual means we do not assign actual wire). If the segment is eventually assigned to a track that is not the same as the track we promote the original terminal to, we just rip-up the original terminal connection and redo it.

In Figure 4.10, we show how we promote the old terminal to upper layer. The horizontal tracks are current layer routing tracks and the dotted tracks are tracks on the upper layer. The vertical short lines in light color are potential via location. The original terminal is located on track $t1$. To promote the old terminal, we extend it with a short wire on track $t1$ and then add the via. The new terminal is shown on the tracks of upper layer.

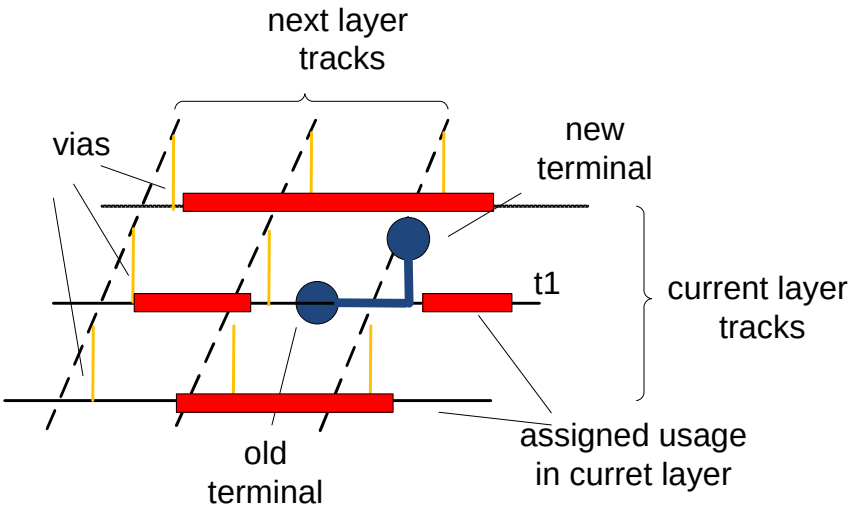

Figure 4.10 Terminal promotion to avoid terminal connection failure.

We could also incorporate the terminal promotion into the MWIS problem. For the segments with incident terminals, choices (vertices) representing terminal promotion is introduced to defer assignment of a segment. We need be aware that the segment should be either assigned to tracks in current layer or its incident terminals be promoted to upper layers. One of the two cases must be met. Hence the weight of the vertex need be set wisely such that choice of exactly one case is taken. This method is doable and we have tried it inside the MWIS problem. But the QOR in terms of routability (i.e., the number of assigned segments) is similar to keeping terminal promotion outside MWIS. Hence we apply terminal promotion as a postprocessing after partial assignment. Overall, the terminal promotion is a highly effective technique in RegularRoute. A valid detailed routing solution is generated after global segment assignment.

Unassigned Segments on Top Layers

For unassigned segments on the top two layers (top two layers with horizontal and vertical tracks respectively), there are no upper layers where we can defer the assignment to. For better routability, we apply panel merging and eventually maze routing if the testcase is too hard.

All of our discussion has been based on the assumption that each segment respects the global routing solution. More specifically, the input global routing solution determines which panel each segment must be assigned. This assumption is restrictive in the fact that it forbids the option of trying alternative panel for better routability.

More specifically, for the panel with unassigned segments, we try to merge it with neighboring few panels and redo the assignment. In the case when the neighboring panel has space for holding more segments, it is likely the problem can be resolved. In the experiment, we merge the neighboring one panel (total three panels). This number can be modified depends on the level of hardness and runtime. The panel merging technique is effective for the segments with preferred tracks near the panel boundary. The merging of panels eliminate the boundary and the segment becomes more flexible.

For the very hard testcases, we could apply the panel merging in lower layers instead of waiting till the top layer. Actually, we could run RegularRoute initially for estimation and

record the panels that are congested (with unassigned segments). We then run RegularRoute again starting from scratch with the precaution of congested panels. The panels that are predicted congested could be merged with neighboring panels since the lower layers. Again, it is a trade-off between solution quality and runtime and we adopt this idea in the case when testcase is too difficult to handle.

If there are still unassigned segment left, we eventually resort to a line probe maze routing technique or a full 3-D maze routing technique. The maze routing is most flexible technique but detour-prone and time-consuming. We adopt it as the last effort in RegularRoute.

Besides maze routing technique, we could also try academic MWIS solver in order to better solve the problem. However, the near-optimal solver such as [48] will be slow in nature. In order to maintain the fast runtime, we keep the fast heuristic as the main solver.

Results and Discussion

All our experiments are performed on a machine with 2.67GHz Intel Xeon CPU and 32G memory. We derive two sets of detailed routing testcases from ISPD98[45] and ISPD05/06 [21, 22] placement benchmark suites respectively. In original ISPD98 placement benchmarks, pins are set to be at the center of each standard cell, we develop a program to randomly set the pin coordinate and make sure they satisfy the spacing requirement at the bottom layer. The size of each module in the derived testcases is the same to that of the IBMv2[49] placement benchmarks. We use Dragon [33] to generate the placed testcases for ISPD98 benchmarks and FastPlace3.1 [39] to place the ISPD05/06 benchmarks. We derive the global routing testcases similar to the format defined by ISPD07/08 global routing benchmarks [28, 29]. We then use FastRoute 4.0[42] to route the global routing testcases and generate the 2-D global routing solution. Both the global routing testcase and the 2-D solution are imported into Regular-Route. Due to the lack of available academic detailed routers, we compared our results with an industrial router - WROUTE. However, WROUTE does not recognize bookshelf placement format or the global routing testcase format we use. We therefore convert the placed testcases by publicly available conversion tool to LEF/DEF format testcases which we can import into

WROUTE. Although the testcases are in different formats [1] We make sure the basic information such as pitch size, module size, routing region, routing layers etc. are identical for both testcases. For the ISPD05/06 placement benchmarks, they are initially proposed for the contest purpose and are not easy to route given the original pitch size (based on the pitch size defined in the ISPD07/08 global routing contest). We thus reduce the pitch size by half and increase the layer count for some testcases.

Results of Local Net Routing

We first show RegularRoute's performance on handling local nets based on the single trunk V-Tree topology. We report the final unassigned local nets, total CPU time, final metal2 usage and unassigned global segment count. Here we only use metal1 and metal2. We compare our results with RSMT topology.

In Table 4.1, the first column lists all experimental testcases. Due to limited space, we only show the results for the ISPD98 derived testcases. The next column shows the total number of local nets for each testcase. The following six columns show results of single trunk V-Tree and RSMT respectively. The RSMT is generated by FLUTE[26] using default settings. First, $\#un.L.$ is the final unassigned local nets. Single trunk V-Tree has no unassigned local nets. But RSMT tree incurs some unassigned nets. Second, CPU is the runtime in seconds. FLUTE runs faster than our algorithm. But the local nets routing runtime is trivial compared with global segment assignment. So the runtime advantage is not important. Third, $metal2$ $usage$ is the total usage on metal2 after routing local nets. The single trunk V-Tree introduces 20% to 30% less metal2 usage, which saves more resources on metal2. $\#un.G.$ is the final unassigned global segment if either topology is applied. RSMT may incur some unassigned global segments and it further suggests RSMT is inferior in preserving routing resources.

[1]The input to RegularRoute is global routing testcases and solution derived similar to that of ISPD07/08 global routing benchmarks [28, 29] and the input to WROUTE is LEF/DEF format design.

Name	#Loc. Nets	Single Trunk V-Tree				RSMT			
		#un. L.	CPU (sec.)	metal2 usage(×10e3)	#un. G.	#un. L.	CPU (sec.)	metal2 usage(×10e3)	#un. G.
ibm01	1081	0	0.04	6.3	0	0	0.02	9.6	0
ibm02	1750	0	0.09	12.8	0	0	0.04	15.3	0
ibm07	4479	0	0.18	22.3	0	7	0.05	32.6	5
ibm08	5539	0	0.23	27.8	0	0	0.11	39.6	0
ibm09	5429	0	0.20	28.2	0	9	0.08	37.9	0
ibm10	2984	0	0.27	17.4	0	0	0.12	29.4	1
ibm11	6983	0	0.26	38.9	0	4	0.07	50.1	7
ibm12	2433	0	0.32	14.5	0	0	0.12	26.8	0

Table 4.1 Results for local net routing on ISPD98 testcases

Global Segment Assignment Results for ISPD98 Testcases

In Table 4.2, we show the results for global segment assignment of RegularRoute on the eight ISPD98 testcases. We compare the results with WROUTE (version 3.0.61). The testcase statistics are shown in the first seven columns, for total number of nets ($\#Nets$), G-Cell grids ($Grid$), total number of global segments ($\#Seg.$), total number of local nets ($\#Loc.Nets$), the average net degree for the whole netlist ($Avg.Deg.$), maximum number of segment ($MaxSeg.$) in one panel and maximum number of pin in one panel ($MaxPin$) respectively. These statistics provide an overall idea about the complexity of these testcases. The next column shows the runtime for FastRoute 4.0[42]. The global routing runtime gives the rough idea of how fast our detailed router compared with the the global router. The following columns show the results of RegularRoute and WROUTE respectively. $\#unassigned$ is the count of segments that cannot be handled by RegularRoute. CPU is the runtime in seconds. The WROUTE results are reported with similar metrics. "$viol.$" is the number of design rule violations, and "$\#fail$" is the number of nets that cannot be connected.

From the table, RegularRoute outperforms WROUTE in both quality and runtime. First, RegularRoute is capable of routing through all the eight testcases. WROUTE, nevertheless, cannot route any testcase without violation. Usually it incurs thousands of DRC violations and a number of failed nets. Second, in terms of runtime, RegularRoute is much better compared

with WROUTE (we bucket the maximum rip-up and reroute rounds, otherwise WROUTE could run for ever), which spends a lot of runtime on rip-up and reroute. On the other hand, it is likely that WROUTE incorporates more design objectives than ours, though we strived to turn off all non-relevant design objectives. However, the point we want to demonstrate is the restrictive regular routing is doable for good routing completion. As we have mentioned in earlier part, we could incorporate more design metrics into our framework. The weight function or cost function for solving MWIS problem can be thus extended to incorporate other design objectives. And we also see better chance for satisfying various design rules, as more and more complicated design rules are triggered by non-trivial routing patterns. The good routing completion rate could also save additional effort during the design rule clean-up stage and thus better manufacturability.

Global Segment Assignment Results for ISPD05/06 Testcases

We show the complete results on ten testcases derived from ISPD05/06 [21, 22] placement benchmarks. They are indeed much bigger in size and more challenging than the ISPD98 derived testcases. We only show the results for ten testcases (sixteen benchmarks in total) because some testcases cannot be routed by WROUTE (either crashes during execution or incurs input constraints violation). As mentioned earlier, these testcases are made following the similar procedure of ISPD98 testcases. We use FastPlace 3.1[39] to place all the placement benchmarks with default setting. Likewise, the results are also compared with WROUTE. We show the number of unassigned segments($\#unassigned$), the CPU time, the via count and total wirelength for RegularRoute and WROUTE. We also show the violation ($viol.$) count for WROUTE. First, RegularRoute is capable of routing all local nets (not shown in Table 3) and assigning all global segments. While WROUTE is likely to incur several failed nets and plenty of design violations. Like the experiments for ISPD98 testcases, we tried our best to switch off other design objectives. Second, Regularroute has smaller via count and wirelength than WROUTE. In particular, smaller via count and wirelength will benefit timing, signal integrity and power consumption. Based on the results, we show RegularRoute is also doing well for

Testcase Statistics							FR4.0	RegularRoute			WROUTE (Encounter)			
Name	#Nets	Grid	#Seg.	#Loc. Nets	Avg. Deg.	Max Seg.	Max Pin	CPU (sec.)	#unassigned	CPU (sec.)	via ×10e5	wlen ×10e5	#fail	viol.	CPU (sec.)	via ×10e6	wlen ×10e7
ibm01	11507	133×132	42307	1118	3.85	238	463	0.47	0	2.69	1.1	6.8	0	1905	145	1.2	6.9
ibm02	18427	152×151	80891	1616	4.23	366	616	2.71	0	11.5	3.4	18.8	0	3266	355	3.5	19.2
ibm07	44394	229×228	162009	5691	3.7	507	877	8.51	0	28.2	5.3	39.9	3	6723	553	5.6	41.8
ibm08	47944	239×238	198188	6905	4.13	568	1042	10.1	0	48.2	6.0	45.9	0	5645	682	6.1	47.8
ibm09	50393	243×242	179942	6590	3.73	509	1016	6.11	0	38.1	4.9	35.0	0	1965	772	4.9	35.5
ibm10	64227	316×315	282041	4329	4.19	640	1045	8.97	0	57.2	8.2	68.5	8	18811	1090	8.5	71.0
ibm11	66994	276×275	230365	8486	3.54	637	1140	15.7	0	55.1	7.0	53.2	0	2716	1112	7.2	54.4
ibm12	67739	341×340	336106	3810	4.34	736	1151	25.4	0	77.3	10.4	98.4	6	21539	1309	11.0	100.5

Table 4.2 Results of RegularRoute and WROUTE for ISPD98 testcases

Testcase Statistics							FR4.0	RegularRoute			WROUTE (Encounter)			
Name	#Nets	Grid	#Seg.	#Loc. Nets	Avg. Deg.	Max Seg.	Max Pin	CPU (sec.)	#unassigned	CPU (sec.)	via ×10e6	wlen ×10e7	#fail	viol.	CPU (sec.)	via ×10e6	wlen ×10e7
adaptec1	219243	893×892	988418	54374	4.28	14242	594	141	0	537	1.9	9.0	0	69337	2401	2.1	9.3
adaptec2	257659	1174×1172	1040019	44356	4.09	15333	065	189	0	469	2.2	10.4	0	58317	3321	2.5	11.0
adaptec3	466293	1935×1946	1887820	44356	4.01	21424	950	342	0	1176	4.0	23.8	5	174295	7939	4.3	24.7
adaptec4	515300	1933×1945	1812333	85000	3.70	18843	820	289	0	1220	3.5	20.8	15	203783	10824	3.9	22.2
adaptec5	867344	1935×1946	3506216	135795	3.99	22034	518	698	0	3607	7.3	48.8	8	222974	13729	7.9	50.3
newblue1	331106	934×932	1070792	69300	3.68	14422	957	42	0	297	2.3	8.8	0	58809	2614	2.4	9.1
newblue5	1257334	2122×2132	4515965	238712	3.87	25215	957	702	0	2654	7.2	46.3	10	628785	14597	7.8	48.8
newblue6	1286448	2310×2318	4944944	208903	4.09	27185	238	598	0	3445	8.5	39.9	0	615376	13645	9.0	41.2
bigblue1	282399	893×892	1182506	25288	4.02	14102	534	134	0	805	2.5	10.9	0	59556	4738	2.7	11.4
bigblue2	576618	1560×1568	1826150	92945	3.60	16483	804	249	0	967	4.2	23.0	4	288271	8856	4.6	23.5

Table 4.3 Results of RegularRoute and WROUTE for ISPD05/06 testcases

modern large designs.

Conclusion

In this paper, we propose a detailed router which seeks to route global segments with regular routing patterns. The whole algorithm is based on a bottom-up layer-by-layer processing. The problem for each layer is partitioned into subproblems by panels. Inside each panel, the global segment assignment is formulated as a MWIS problem. An effective heuristic and a few postprocessing techniques are developed. We have shown RegularRoute's performance on two sets of detailed routing testcases. In the future, we would like to further improve RegularRoute's performance and incorporate more design objectives to make our tool more suitable for industrial applications. In addition, we are interested in making a parallel version of our tool for further runtime reduction.

Acknowledgments

The authors would like to thank Dr. Hardy Leung for valuable discussions which inspire our work, and the University of Michigan CAD group for the helpful placement utility tools to convert bookshelf placement format to LEF/DEF format.

CHAPTER 5. IGD: An Integrated Global Routing and Detailed Routing Algorithm

A paper submitted to *ACM/IEEE Design Automation Conference*

Yanheng Zhang and Chris Chu

Abstract

Conventional ideas divide VLSI routing into global routing and detailed routing for maneuvering large design data. In global routing, routability associated evaluation is abstracted by the global routing capacity for each global edge. However, the global capacity model is either too simplistic or too optimistic to reflect real routing hurdles in detailed routing.

In this paper, we propose IGD: a novel integrated global routing and detailed routing approach for remedying the inconsistency between global routing and detailed routing. Before global routing, our approach starts with a more accurate initial capacity prediction for all global edges. The initial capacity prediction model considers local pins, local connections and probabilistic pin promotion inside each 3-D global cell (i.e., 3D Cell). Then we use an effective iterative flow for improving routability for detailed routing in terms of number of unassigned segments. The hotspot in detailed routing in current iteration is fed back to the global routing stage to adjust the global capacity in the previous round. To ensure computational feasibility, fast and efficient academic global routing and detailed routing algorithms *FastRoute*, *RegularRoute* are integrated. We also propose some well-designed methods to achieve short runtime while maintaining solution quality. Experimental results reveal that our algorithm surpasses conventional routing flow in routability. In particular, it can reduce the unassigned segments

by 80% on average on ISPD98 derived testcases with roughly 3 × runtime overhead compared with traditional flow.

Introduction

VLSI routing consists the task of connecting pins and ports using over-the-cell metal wires based on logical connection defined by netlist. The routing performance directly impacts design metrics such as interconnect delay, power consumption and chip reliability etc. As the fabrication technology enters the nanometer era, VLSI routing is becoming increasingly important. In one aspect, the degree of complexity of design is increasing dramatically as more modules are integrated onto the chip. The routing requirement for connecting the pins and ports of these modules is much higher. In another aspect, with diminishing feature size, there are more complex design rules imposed to ensure manufacturability. It has been reported that for 32nm process, the number of rules reaches several thousands [1] and the design rule manual has roughly a thousand pages [2]. The routing closure (i.e., generate DRC free routing solution) is harder to achieve and will eventually affect chip time to market (TTM) plan.

It has been proved that VLSI routing is NP-hard. To manage large design data, conventional flow usually divides the whole routing procedure into global routing and detailed routing stages. In particular, in global routing stage, layout region is partitioned into G-Cells (i.e., global cells) and rough routing decision is made based on G-Cell to G-Cell connection on a global routing grid graph. The subsequent detailed routing realizes exact routing path of the global routing solution considering both geometrical constraints for metal wires as well as various design rules.

	Global Routing		Detailed Routing	
Name	O.F.	CPU(sec.)	unassiged	CPU(sec.)
adaptec1	0	195	17956	566
adaptec2	0	48	34372	442
adaptec3	0	324	28549	1285
adaptec4	0	66	30772	1330
adaptec5	0	559	69017	3782

Table 5.1 Detailed routing solution by detailed router RegularRoute for routable global routing benchmarks in ISPD07/08 global routing contest.

The common objective for global routing is to generate a congestion free solution where the wiring demands across the G-Cell is below its capacity, or global edge capacty. There have been many research conducted for developing high performance global router to resist congestion with given global edge capacity. As revealed by two consecutive annual global routing contests hosted by ISPD, sequential ripup and reroute based approach is in dominance over concurrent approach. Contest winning routers: BoxRouter[6], NTHU-R[12], FGR [11] and FastRotue3.0[50] are proposed during the period of the two contests. Although these routers acomplish the task of obtaining congestion free solution for most routers, the actual routability of subsequent detailed routing stage is pending with doubt. In Table 5.1, we compare the detailed routing results using an academic detailed router for five routable benchmarks of the contest. The global routing solution is generated using FastRoute 4.0 [42]. The results show that none of the global routing solution can be routed with the contest defined pitch size. Although these global routing benchmarks are routable in global routing stage, it is hard to gurantee the same success in detailed routing stage. In other words, the utilized global edge capacity in these benchmarks underestimate the hardness of subsequent detailed routing stage. It is a weak indicator of the real routing hurdles that follows.

Traditionally the value of the capaicty could be roughly determined based on empirical methods. For instance, certain amount of capacity value is deducted for global edges on first horizontal and vertical metal layer. And the capacity of global edges that are covered by macro blocks are subject to scaling called macro porosity. However, these methods usually are not accurate enough as they lack the local information of each G-Cell. Usually it is not easy to make a correct choice. Underestimating the capacity will consume more wirelength while being optimistic might results in trouble for acomplishing routing closure.

In this paper we will present a novel algorithm for addressing the inconsistency between global and detailed routing stages. Based on our knowledge, the approach is the first algorithm ever that a detailed router is fully integrated with a global router based on efficient global routing and detailed routing algorithms *FastRoute* and *RegularRoute*. Unlike traditional capacity assignment methods, our algorithm starts with an initial capacity estimation considering local

pins, local nets and probabilistic pin promotions. Then the global routing testcase is routed by *FastRoute* and *RegularRoute*. Based on congestion information generated by detailed router of current iteration, we adjust the global edge capacity and then retest the testcase with new capacity. This process is iteratively applied for continuous refinement until congestion free detailed routing solution is obtained. To perform full global and detailed routing inside the iterative flow will be very expensive in runtime, we also propose the incremental maze routing and history-based global segment assignment for alleviating computational complexity.

Thus, novel techniques in this paper are listed below:

- A novel framework integrating global routing and detailed routing
- An effecitve initial global edge capacity estimation by local information of each G-Cell
- A useful feedback methodology to adjust the global edge capacity based on detailed routing solution
- An incremental global routing method and history based idea in detailed routing for runtime speedup

We implemented IGD and tested its performance on ISPD98[45] derived testcases. Experimental results show that IGD is capable of generating detailed routing friendly global routing solutions. The detailed routing stage routability in terms of unassigned segments is improved by 80% with roughly 3 $\times$ runtime overhead. This justifies the necessity of integrating detailed routing into global routing stage.

The rest of paper is organized as follows: In Section 2, we first review the efficient global routing and detailed routing algorithms *FastRoute* and *RegularRoute* and present the algorithm flow of IGD. Section 3 discusses the initial global edge capacity prediction. In Section 4, techniques to integrate global routing into detailed routing are discussed in detail. Experimental results are shown in Section 5 and we will make conclusion of this paper in Section 6.

Materials and Methods

IGD Overview

In this section, we will first present the problem formulation and definitions for global routing and detailed routing respectively. We then review effective global routing and detailed routing algorithms *FastRoute* and *RegularRoute*. Finally, we will present the flow of our integrated algorithm.

Problem Formulation

We will introduce the formulation for global routing and detailed routing respectively. In global routing stage, layout region on each layer is divided into a set of 3-D global routing cells (3-D Cells). The 3-D global routing grid graph is built where each grid point denotes one 3-D Cell and each global edge represents the common boundary between two 3-D Cells. Each global edge is assigned with a capacity representing the allowed number of routing usage. And overflow is the exceeding value of routing usage over global edge capacity. In other word, $o_e = u_e - c_e$ where o_e, u_e and c_e represent overflow, usage and capacity of edge e respectively. The primary objective in global routing stage is to minimize overflow. In *FastRoute*, we lump the 3-D G-Cells with same X's and Y's coordinates on different layers into a 2-D G-Cell (or G-Cell). The capacity, usage and overflow associated global edges will be lumped together as well to become the capacity, usage and overflow of the 2-D global edge respectively. The 3-D grid graph is projected to become a 2-D grid graph. The 2-D solution is first generated and 3-D can be obtained by layer assignment by the 2-D solution.

The detailed routing is formulated the same as *RegularRoute*. The routing resource is modeled as a 3-D regular grid graph. Each grid can accomodate one wire except for edges with blockage. Each layer of graph has a *preferred routing direction* and the preferred direction of adjacent layers are perpendicular to each other. And we assume the preferred direction of lowest layer (metal1) is horizontal. A sequence of unblocked grid edges along the preferred routing direction of each layer is called *routing track*. The input is a placed netlist with exact

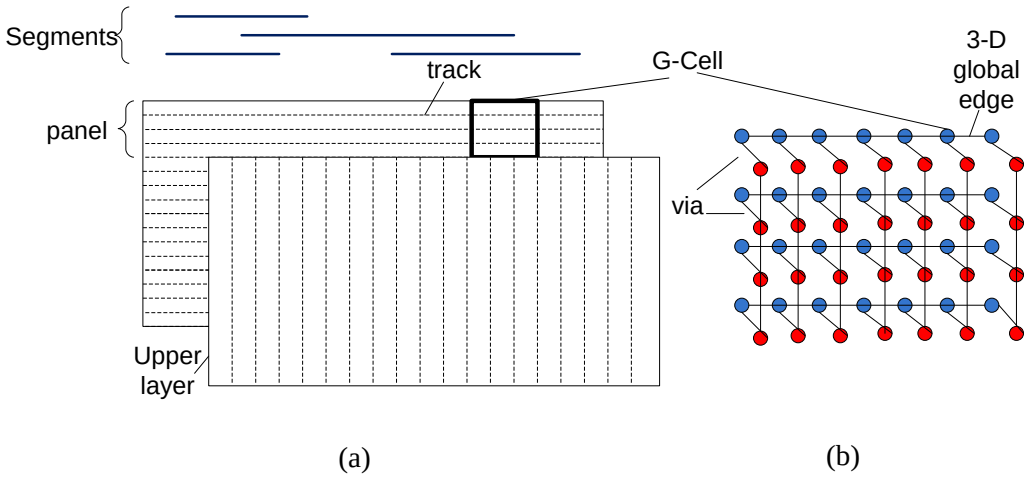

Figure 5.1 Problem formulation for global routing and detailed routing (a) Detailed routing with panel, track, segments with two layers (b) Corresponding global routing grid graph

pin locations and a corresponding 2-D global routing solution in labyrinth[51] format. The segments are extracted from the 2-D global routing solution representing the horizontal or vertical interval spanning multiple G-Cells. We also define *panel* to be the collection of all tracks on one layer within one row (for odd layer) or one column (for even layer) of G-Cells. If we respect the input 2-D global routing solution, the segment will be assigned to the tracks on a deck of panels that are on different layers but associated with the same row/column of G-Cells. The detailed routing is to route all nets (route all local nets and assign all extracted global segments) on the grid based on the 2-D global routing solution such that routes of different nets do not intersect.

The formulation of global routing and detailed routing is illustrated in Figure 5.1. It shows the concept of G-Cell, global routing grid graph, global routing edge, panel, routing track and segment respectively.

FastRoute and RegularRoute

FastRoute is very fast and effective global router. It incorporates many novel ideas for solving increasingly complicated testcases. The work contains a series of publications in main EDA

conferences. [7] is the initial work which includes congestion-driven steiner tree construction and edge shifting. Later in [8], a novel monotonic routing and multi-source multi-sink maze routing ideas are proposed to enhance the pattern routing and traditional maze routing. In [50], a virtual capacity idea is introduced for better convergence in maze routing stage. And the latest works [42, 52] introduce novel ideas for reducing via count and routing detour. Experimental results have shown that FastRoute is able to generate state-of-the-art global routing solutions.

RegularRoute[53] is a recently proposed grid-based detailed routing technique. It uses a bottom-up layer by layer strategy to solve the problem. It employs a single trunk V-Tree technique to route the local nets. The global segments are assigned panel by panel and inside each panel a MIS problem is formulated. After solving the MIS problem with efficient heuristic, some postprocessing techniques are proposed to improve assignment rate. In addition to good assignment rate, RegularRoute is very fast academic detailed router which enables the possibility of integrating it inside IGD.

Algorithm Flow

The proposed flow is shown in Figure 5.2. Basically the flow contains three stages: (1) global capacity prediction, (2) initial routing, and (3) iterative improvement. The flow of IGD incorporates the flow of *FastRoute* and *RegularRoute* but it is not simply an addition of the two. There are many new techniques introduced due to the new objective is to generate best detailed routing solution.

In particular, the first stage predicts the global capacity for all global edges. Two techniques are involved. One is a conservative reduction based on local information of adjacent 3D Cells for first two layers (H and V). The second is a probabilistic methodology for predicting the promotion of some pins and capacity of related global edges is further reduced. Stage2 performs initial global routing and detailed routing using the standard *Fastroute* and *RegularRoute* flow. After this stage, an iterative improvement is applied to improve the number of unassigned segments in previous round. We propose adaptive capacity adjustment scheme for adjusting

Stage1: Global Capacity Estimation

2. Generate single trunk V-Tree for local nets
3. Conservative capacity reduction of first two layers
4. Probabilistic pin promotion and further capacity reduction

Stage2: Initial Routing

5. Congestion-driven steiner tree construction
6. Pattern routing in "L" and "Z" shape
7. Repeat
   a. Full maze routing in rip-up and reroute
8. Until obtain congestion free global routing solution
9. Segment extraction and organization
11. Route global nets by assigning all global segments
12. Obtain information of unassigned segments

Stage3: Iterative Improvement

13. Global capacity adjustment based on unassigned segments
14. Repeat
   a. Incremental maze routing in rip-up and reroute
15. Until congestion no longer improves
16. Segment extraction and organization for nets rerouted in incremental maze routing
17. Assign global segments by history-based technique
18. Repeat Stage 3 until number of unassigned segments stops improving

Figure 5.2 Flow of IGD

the global capacity of previous round. We will then re-run the routing test. To speedup the algorithm, we introduce an incremental maze routing method and a history based global segment assignment technique. Moreover, in IGD, detailed routing has been fully integrated in the flow. Eventually the output is not only a detailed routing friendly global routing solution, but also the detailed routing solution over it.

Global Capacity Prediction

In this section, we will introduce the global capacity prediction in detail. In the first part, we will discuss a conservative capacity reduction based on local information of 3-D Cells in first horizontal metal layer (metal1) and first vertical metal layer (metal2). Then in the second part we will present a novel probabilistic pin promotion technique to reduce the capacity for 3-D Cells for layers above metal1.

Conservative Capacity Reduction

In global routing, determining the global capacity for each global edge plays an important role in guaranteeing good solution. In *FastRoute*, we lump the capacity of all 3-D global edges as the capacity for the designated 2-D global edge. To determine the capacity for a 3-D global edge e, the easiest method of calculating the *full capacity* (e_{FC}). We count the available number of tracks across the common boundary of the two 3-D Cells (length of 3-D G-Cell boundary divided by pitch). However, it is an overestimation of the capacity as there are local blockages such as pins and local connections which will block some tracks. The resulting global routing solution can be hard to route in detailed routing as the capacity are defined to be too optimistic. To solve this problem, some empirical methods proposed to reduce certain amount of capacity out of the full capacity of each 3-D global edge such as the way people use in ISPD07/08 global routing contest[28, 29]. Again, this idea is hard to manipulate. It is necessary to tune the amount of reduction for different set of testcases which is very time-consuming.

We propose *conservative capacity reduction*(CCR) technique to tackle this problem. We initially calculate the full capacity for all global edges. Then we look into the associated two

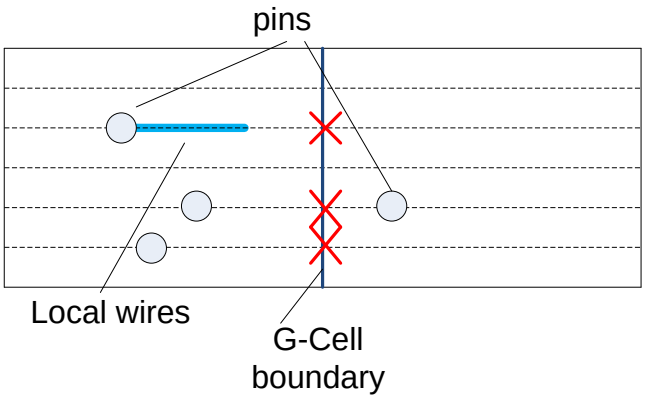

Figure 5.3 Conservative capacity reduction

3-D Cells of each global edge, we analyze the local pin and local connections that reside on the tracks of the two 3-D Cells. We call those tracks as *potential blocked tracks*(PBT). The capacity is reduced by the number of PBT from full capacity.

Pins are easier to consider because they are all by themselves. For local nets that need be connected, we use the single trunk V-Tree (V-Tree) method as in [53] only using metal1 and metal2. The local connections will be fixed and regarded as blockages on certain routing tracks.

Figure 5.3 denotes one horizontal global edge and the two related 3-D Cells on metal1. The full capacity of the edge is 5, and the number of PBT is 3. There are two PBTs caused by pins and one caused by local connection. Then the capacity for this global edge will be adjusted to be 2. We apply this idea for all global edges on metal1 and metal2 because all pins are located on metal1 and we only use metal1 and metal2 to construct the local connection[1].

Probabilistic Pin Promotion

In *RegularRoute*, global segments are assigned based on a layer by layer manner. To prevent the disconnection between the segments that are assigned in upper layers and their pins on metal1. We promote the pins to upper layer after processing the segments in layer. The promoted pins are then regarded as local blockages for 3-D Cells in the upper layer.

[1]Based on the experimental results we have on all ISPD98 derived benchmarks, all local nets can be routed using the first horizontal and vertical metal layer.

However, without exact detailed routing, we are unable to know which pins need be promoted and how they will affect the global capacity in the upper layer. Although it's doable for performing a testing round of detailed routing, it will affect the runtime budget of our tool. We therefore propose a *probabilistic pin promotion* (PPP) technique assisting estimating the pin promotion effect on global capacity for global edges above metal1.

The PPP technique is performed in a bottom-up layer by layer flow. For each 3-D Cell on current layer, we evaluate its *pin density* based on the ratio of used routing resouce (grid usage) and total routing resource inside a user-defined window. Note that the pin density value should be between 0 and 1 ([0,1]). 0 means no usage or blockage surrounding the pin, while 1 means all the grid points inside the window have been taken up. We then generate a random number between 0 and 1 for the pin. If the number is below the ratio, we will assume it will be promoted to upper layer. For instance, if one pin is located in metal1, its pin density is 0.5 and the random number is 0.4, then it is promoted to metal2. The smaller the pin density, the less possible it will be brought. The reason for generating the random number instead of using the pin density directly is to avoid the case when the pins are quite dense in some region, all pins in that region will be promoted. But based on our experiments, even for the regions with high pin density, many pins can be routed without promotion.

The idea is illustrated in Figure 5.4. The box is used to evaluate the density surrounding the pin, which is 0.375. If the generated random number is smaller than 0.375, it will be assumed to be promoted. The promoted pin will be on the next layer with the same X's and Y's coordinate. We continue this process until reaching the toplayer or when no more pins can be promoted. For the 3-D Cells with promoted pin above metal1, we further reduce their capacity based on CCR as discussed before.

Actually the CCR and PPP techniques work on making a initial estimation of the capacity using the local information for all 3-D Cells. Is is more accurate than convential ideas which usually apply uniform capacity based on some empirical methods. Our new techniques have been verified by experimental results which will be presented in Section 5.

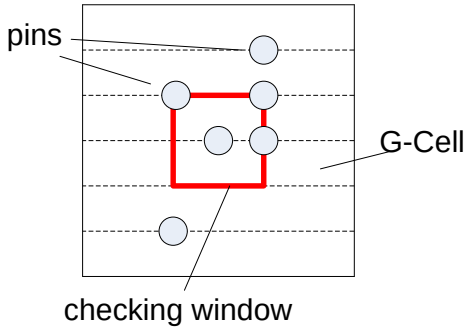

Figure 5.4 Probabilistic pin promotion using window based pin density evaluation

Iterative Improvement Framework

In this section we will talk about *iterative improvement*, the major idea is to provide feedbacks based on the detailed routing information in current round. The unrouted segment in detailed routing stage will be used to adjust the capacity of global edges. And then we perform incremental global routing and a novel history based detailed routing to incrementally improve the detailed routing solution QoR in terms of unassigned segments. Finally we will obtain not only a detailed routing friendly global routing solution, but also a detailed routing solution over it.

Adaptive Global Capacity Adjustment

As the flow that is shown in Figure 5.2, after the initial capacity prediction, we will perform initial global routing and detailed routing using *FastRoute* and *RegularRoute* with their default settings. During the global routing stage, the 2-D global routing solution in terms of the routing path on the global routing grid graph will be obtained. We will then organize the solution into a set of global segments and then assign the segments in the subsequent detailed routing step.

The routability information in terms of the unassigned segments will indicate where the possible detailed routing hotspots are. Although we have made initial "guess" about the global capacity for each 3-D G-Cell in the initial capacity estimation stage, they are far from being precise.

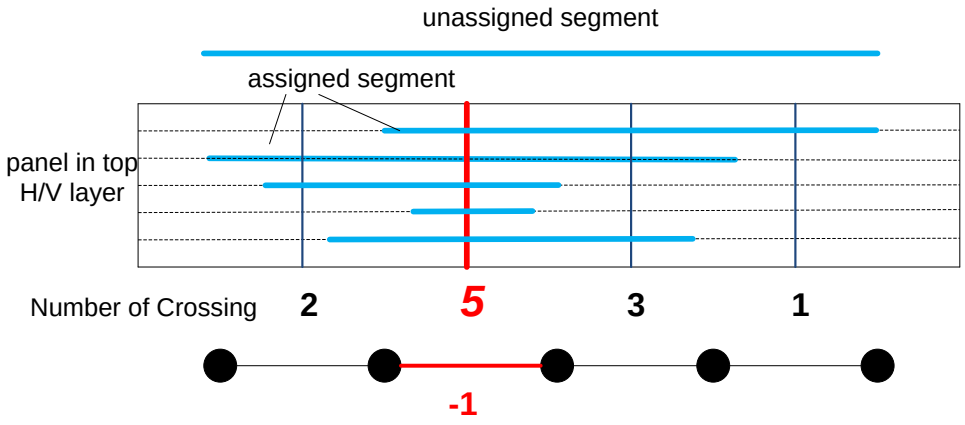

Figure 5.5 Adaptive global capacity adjustment based on unassigned global segment

As defined in Section 2.1, a segment in detailed routing is an interval spanning multiple G-Cells. If we respect the global routing solution, the segment should be assigned to the tracks on a deck of panels that are on different layers but associated with the same row/column of 3-D G-Cells. If a segment is unassigned, it indicates there's no room to accomodate the assignment inside the corresponding panel on the top layer (top horizontal or vertical metal layer depending on the direction of segment).

For the unassigned segment, we will investigate the panel on the top layer the segment resides and find the congested spots. We define the congestion based on the number of crossing for all global edges the segment goes through. The crossing means the already assigned segment across one particular global edge. As illustrated in Figure 5.5, the unassigned segment spans 5 3-D G-Cells on the top horizontal layer and the hotspot is the global edge with most number of crossing, which is second G-Cell boundary (5 corssings). Then the capacity of the corresponding global edge is reduced by some amount (we use 1). The capacity is reduced such that some routing usage need be pushed out. We didn't reduce the capacity much is (1) we are still not very certain which global edge is the "real" bottleneck, it is still a in a testing phase (2) we do not want to disturb the original solution too much, which would lead to much more effort in global routing and detailed routing in the following iteration. For the global edges where the difference between usage and capacity is over a threshold ($c_e - u_e > t$, where t is the threshold, and u_e will be equal to the number of crossing), we will increase their capacity (c_e) by some

amount but not exceed the full capacity to absort more usage. This method can spread the usage more evenly and help alleviate the congestion.

Methods for Runtime Consideration

Up till now, although the proposed idea is feasible, yet the runtime will be a major concern if we iteratively apply full global routing and detailed routing. As a matter of fact, with smaller and smaller capacity value on some of the global edges, the global routing might become too congested to acomplish convergence. And it will spend much more runtime after a few iterations. To avoid performing routing from scratch, we propose speedup techniques for global routing step and detailed routing step inside the iterative improvement flow respectively.

Incremental Maze Routing Instead of applying a full global routing after the adaptive capacity adjustment, we perform incremental maze routing. We apply the maze routing for the nets that incur congestion after capacity adjustment. The maze routing might not be able to produce overflow-free solution. In that case, we will terminal the rip-up and reroute and accept the best available result. After we get the incremental maze routing solution, we will re-extract the global segments only for the nets that are rerouted during incremental maze routing. It saves runtime for extracting and organizing all segments.

History-based Detailed Routing In *RegularRoute*, solution space of assigning global segment is explored based on the choice of the segment. Basically the choice defines the components that consist one regular routing solution for the segment. It specifies the particular track, layer, and terminal connection options. The more choices provided for a segment, the more flexible the segment can be assigned. However, as noted before, to perform a full detailed routing inside the iterative flow will be quite expensive. The history-based idea can compensate the runtime loss while maintains good solution quality. For the nets that are not rerouted and the derived global segments are successfully assigned last iteration, instead of generating the whole pool of choices, we will recommend the choice the segment was assigned in the previous round (rank the recommended choice with higher priority). If the segment is consistently

assignable, we gradually decrease the number of choice of it. Otherwise, for the segments that are tough to handle, we increase the number of choice or even provide full flexibility (incorporate all possible choices).

Results and Discussion

We implement IGD in C and all our experiments are performed on a machine with 2.67GHz Intel Xeon CPU and 32G memory. We derive the testcases from ISPD98[20] similar to the work in [53]. In original ISPD98 placement benchmarks, pins are set to be at the center of each standard cell, we develop a program to randomly set the pin coordinate and make sure they satisfy the spacing requirement at the bottom layer. The size of each module in the derived testcases is the same to that of the IBMv2[49] placement benchmarks. We use Dragon[33] to generate the placed testcases for ISPD98 benchmarks. After obtaining the placed testcases, we derive the global routing benchmark with the same format as ISPD07/08 global routing contest benchmarks [28, 29].

Initial Capacity Prediction Results

	empirical		ours	
Name	unassigned	CPU(sec.)	unassigned	CPU(sec.)
ibm01	0	4.14	0	4.49
ibm02	66	9.58	45	11.3
ibm07	94	24.3	63	26.5
ibm08	53	35.2	32	37.7
ibm09	72	32.9	27	35.0
ibm10	95	59.5	70	63.9
ibm11	79	41.0	22	43.5
ibm12	264	74.5	180	77.9
Sum	723	281.1	459	300.3
Norm	1.58	1	1	1.067

Table 5.2 Results comparison with empirical capacity assignment up to stage2 in IGD on ISPD98 derived testcases.

We will first show the performance of initial capacity prediction on ISPD98 derived testcases.

The results are summarized in Table 5.2. The results are compared against empirical methods. The empirical method assumes the lumped 2-D global edge capacity to be 80% of the full capacity. For demonstrating the effectiveness of the idea, we only run till stage2 which do not employ the iterative improvement. The global router (*FastRoute*) and detailed router *RegularRoute* are executed using the same parameter settings.

In Table 5.2, the first column lists the name of all experimental testcases. The next two columns show the unassigned segments and total CPU time evaluated in seconds for empirical methods. The final two columns report the results for IGD. We find that IGD surpasses empirical methods in detailed routing routability, we are able to reduce 60% of unassigned segments in the initial routing stage (stage2). We also observe that the runtime overhead of the capacity prediction is small as we are only 7% slower in the total runtime up to stage2 than the empirical methods. These results suggest that the initial capacity prediction is capable of providing a better start for producing detailed routing friendly solutions.

Full Results on ISPD98 Testcases

In this part, we will show full results for IGD on the eight ISPD98 derived testcases. We compare our results with several modes of IGD. All the results are summarized in Table 5.3. In the table, $\#Nets$ is the total number of nets in the testcase, *Grid* is the scale of global routing grid, *Avg.Deg.* is the average net degree for the whoe netlist. These statistics provide a basic idea of the complexity for each testcase.

The following columns compare the results in more detail on 3 different modes of IGD. In particular, *mode*1 is IGD without initial capacity estimation, *mode*2 is IGD without iterative improvement, *mode*3 is IGD with iterative improvement, but using full global routing and detailed routing instead of the incremental maze routing and history-based global segment assignment technique. The results of IGD is presented in final columns. From the table, we are able to view the necessity of all stages inside IGD's flow. By comparing with *mode*1, we observe that initial capacity is useful in providing a good start. The subsequent refinement becomes more powerful and the final results are also improved in terms of unassigned global

Name	statistics			mode1			mode2			mode3			ours		
	#Nets	Grid	Avg. Deg.	unass-igned	CPU (sec.)	iter	unass-igned	CPU (sec.)	iter.	unass-igned	CPU (sec.)	iter.	unass-igned	CPU (sec.)	iter.
ibm01	11507	133×132	3.85	0	4.49	0	0	4.49	0	0	4.49	0	0	4.49	0
ibm02	18427	152×151	4.23	12	39.4	5	45	11.3	0	5	158.2	4	8	41.5	5
ibm07	44394	229×228	3.70	28	80.2	4	63	26.5	0	11	409.5	4	14	83.4	4
ibm08	47944	239×238	4.13	8	100.4	4	32	37.7	0	0	397.2	4	0	102.8	4
ibm09	50393	243×242	3.73	6	112.1	4	27	35.0	0	0	346.2	5	0	114.8	5
ibm10	64227	316×315	4.19	23	266.7	5	70	63.9	0	11	1207.4	5	11	270.5	5
ibm11	66994	276×275	3.54	0	172.3	4	22	43.5	0	0	968.0	4	0	175.5	4
ibm12	67739	341×340	4.34	69	384.9	5	180	77.9	0	43	1428.6	5	51	388.2	5

Table 5.3 Results comparison for IGD with different modes

segments. *mode*2 suggests the effectiveness of global capacity adjustment. Without the iterative improvement, the detailed routing solution is becoming much worse. *mode*3 indicates that runtime is a concern if we apply full global and detailed routing inside the iterative flow. The detailed routing solution quality is quite similar but they are very expensive and the total runtime is on average 5× slower than IGD.

Conclusion

In this paper, we propose IGD: an integrated global routing and detailed routing algorithm. It is the first algorithm as far as we know to consider detailed routing performance inside global routing. To enable computational feasibility, two high-performance global and detailed routers are integrated. We introduce innovative initial capacity estimation as well as adaptive capacity adjustment based on the detailed routing results. For runtime concerns, we also propose techniques to speedup IGD including incremental maze routing and history-based detailed routing.

In the future, we would like to further improve IGD's performance and scalability. We will need to propose more intelligent schemes to perform capacity adjustment. Meanwhile, we will also try to incorporate better global routing or detailed routing algorithms for better QoR.

CHAPTER 6. GENERAL CONCLUSIONS

General Discussion

In this thesis, we have provided a series of systematic solutions to the three routability related physical design phases for placement, global routing and detailed routing respectively. We have proposed four tools for improving the routability for specific phase or integrating some phases to build a more powerful and generic flow in achieving routing success. In particular, we have shown that FastRoute 3.0 can generate high quality solutions for ISPD98, ISPD07 and ISPD08 global routing benchmarks. The placement refinement tool, CROP is capable of alleviating congestion for both the global routing routability as well as detailed routing routability for various placement tools. The detailed router we propose is capable of producing better results for two sets of detailed routing testcases on both routability and design rule satisfaction. The results of IGD also show the integration of global routing and detailed routing can successfully reduce the number of unassigned segments. From all these results, we can see more potential for achieving better routing QoR for modern VLSI designs.

Recommendations for Future Research

We will continue work on improving the performance and scalability for each tool. For instance, we need further improve the routing wirelength besides the high performance we achieved in runtime and routability for FastRoute. For CROP, we will try to continue improve its scalability for the congestion-driven detailed placement technique. For RegularRoute, we will try to incorporate more design objectives besides routability. We will also make a parallel version of the tool for further runtime speedup. In terms of IGD, we need further analyze

better feedback strategies and try to integrate better global routing and detailed routing tools to improve its efficiency.

We will also make efforts to better remedy the inconsistencies between each phase so that the routability objective can be carried out throughout the design flow. It is even possible for considering the routability issue at early physical design phases or even early design stages such as logical level design.

REFERENCES

[1] Danny Rittman. Nanometer DFM – the tip of the ice. Intelligence: From Science to Industry, Tayden Design newsletter, March 2008.

[2] Luigi Capodieci. Layout printability verification and physical design regularity: Roadmap enablers for the next decade. In *edp*, 2006.

[3] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, NY, 1979.

[4] E. Bozorgzadeh R. Kastner and M. Sarrafzadeh. Pattern routing: Use and theory for increasing predictability and avoiding coupling. *IEEE Trans. on Computer-Aided Design and Integrated Circuits and Systems*, 21(7):777–790, July 2002.

[5] Z.Jiang, B.Su, and Y.Chang. Improved global routing through congestion estimation. In *Proc. ACM/IEEE Design Automation Conf.*, pages 28–31, 2003.

[6] M. Cho and D. Z. Pan. Boxrouter: A new global router based on box expansion and progressive ilp. In *Proc. ACM/IEEE Design Automation Conf.*, pages 373–378, 2006.

[7] M. Pan and C. Chu. Fastroute: A step to integrate global routing into placement. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 464–471, 2006.

[8] M.Pan and C.Chu. FastRoute 2.0: A high-quality and efficient global router. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 250–255, 2007.

[9] M. M. Ozdal and M. D.F. Wong. Archer: A history-driven global routing algorithm. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 481–487, 2007.

[10] K. Yuan M. Cho, K. Lu and D. Z. Pan. Boxrouter 2.0: Architecture and implementation of a hybrid and robust global router. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 503–508, 2007.

[11] M. M. Ozdal and M. D.F. Wong. High-performance routing at the nanometer scale. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 496–502, 2007.

[12] P.-C. Wu J.-R. Gao and T.-C. Wang. A new global router for modern designs. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 232–237, 2008.

[13] A. Hashimoto and J. Stevens. Wire routing by optimizing channel assignment within large apertures. In *Proc. ACM/IEEE Design Automation Conf.*, pages 155–169, 1971.

[14] T. Yoshimura and E. Kuh. Efficient algorithms for channel routing. *IEEE Trans. on Computer-Aided Design*, 1(1):633–647, Jan 1982.

[15] J. Cong, J. Fang, and K. Khoo. DUNE-a multilayer gridless routing system. *IEEE Trans. on Computer-Aided Design*, 20(5):633–647, May 2001.

[16] Y. Chang and S. Lin. MR: A new framework for multilevel full-chip routing. *IEEE Trans. on Computer-Aided Design*, 23(5):793–800, May 2004.

[17] G. Nam, K. Sakallah, and R. Rutenbar. A new FPGA detailed routing approach via search-based boolean satisfiability. *IEEE Trans. on Computer-Aided Design*, 21(6):674–684, Jun 2002.

[18] S. Batterywala, N. Shenoy, W. Nicholls, and H. Zhou. Track assignment: A desirable intermediate step between global routing and detailed routing. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 59–66, 2002.

[19] M. Ozdal. Detailed-routing algorithms for dense pin clusters in integrated circuits. *IEEE Trans. on Computer-Aided Design*, 28(3):340–349, March 2009.

[20] ISPD98 global routing benchmarks. http://www.ece.ucsb.edu/∼kastner/labyrinth.

[21] ISPD05 placement contest benchmarks. http://www.sigda.org/ispd2005/contest.htm.

[22] ISPD06 placement contest benchmarks. http://www.sigda.org/ispd2006/contest.htm.

[23] ISPD08 global routing contest results. http://www.ispd.cc/ispd08_technical_program.html.

[24] M.Pan and C.Chu. FastRoute 2.0: A high-quality and efficient global router. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 250–255, 2007.

[25] L. Mcmurchie and C. Ebeling. Pathfinder: A negotiation-based performance-driven router for fpgas. In *Proc. ACM/SIGDA Intl. Symp. on Field-Programmable Gate*, pages 111–117, 1995.

[26] C. Chu. Flute: Fast lookup table based wirelength estimation technique. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 696–701, 2004.

[27] M. D. Moffitt. Maizerouter: Engineering an effective global router. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 226–231, 2008.

[28] ISPD07 global routing contest benchmarks. http://www.sigda.org/ispd2007/contest.htm.

[29] ISPD08 global routing contest benchmarks. http://www.sigda.org/ispd2008/contest.htm.

[30] P.Spindler and F.M.Johannes. Fast and accurate routing demand estimation for efficient routability-driven placement. In *Proc. Conf. on Design, Automation and Test in Europe*, pages 1226–1231, 2007.

[31] Z.Jiang, B.Su, and Y.Chang. Routability-driven analytical placement by net overlapping removal for large-scale mixed-size designs. In *Proc. ACM/IEEE Design Automation Conf.*, pages 167–172, 2008.

[32] K.Tsota, C.Koh, and V.Balakrishnan. Guiding global placement with wire density. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 212–217, 2008.

[33] X.Yang, B.Choi, and M.Sarrafzadeh. Routability-driven white space allocation for fixed-die standard-cell placement. *IEEE Trans. on Computer-Aided Design and Integrated Circuits and Systems*, 22(4):410–419, April 2003.

[34] C.Li, M.Xie, C.Koh, J.Cong, and P.Madden. Routability-driven placement and white space allocation. *IEEE Trans. on Computer-Aided Design and Integrated Circuits and Systems*, 26(5):167–172, May 2008.

[35] U.Brenner and A.Rohe. An effective congestion-driven placement framework. In *Proc. ACM/SIGDA Intl. Symp. on Physical Design*, pages 6–11, 2002.

[36] M.Pan and C.Chu. IPR: An integrated placement and routing algorithm. In *Proc. ACM/IEEE Design Automation Conf.*, pages 59–62, 2007.

[37] N.Viswanathan and C.Chu. FastPlace: efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model. In *Proc. ACM/SIGDA Intl. Symp. on Physical Design*, pages 26–33, 2004.

[38] J.Roy and I.L.Markov. Seeing the forest and the trees: Steiner wirelength optimization in placement. *IEEE Trans. on Computer-Aided Design and Integrated Circuits and Systems*, 26(4):632–644, April 2007.

[39] N.Viswanathan, M.Pan, and C.Chu. FastPlace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 135–140, 2007.

[40] T.Chen, Z.Jiang, T.Hsu, H.Chen, and Y.Chang. NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *IEEE Trans. on Computer-Aided Design and Integrated Circuits and Systems*, 27(7):1228–1240, July 2008.

[41] T.F.Chan, J.Cong, M.Romesis, J.R.Shinnerl, K.Sze, and M.Xie. mPL6: a robust multilevel mixed-size placement engine. In *Proc. ACM/SIGDA Intl. Symp. on Physical Design*, pages 227–229, 2005.

[42] Y.Xu, Y.Zhang, and C.Chu. FastRoute 4.0: Global router with efficient via minimization. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 576–581, 2009.

[43] M.Pan, N.Viswanathan, and C.Chu. An efficient and effective detailed placement algorithm. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 48–55, 2005.

[44] Y.Chang, Y.Lee, and T.Wang. NTHU-Route 2.0: A fast and stable global router. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 338–343, 2008.

[45] IBM-Place 1.0 benchmark suites. http://er.cs.ucla.edu/benchmarks/ibm-place/.

[46] H. Shin and A. Vincentelli. A detailed router based on incremental routing modifications: Mighty. *IEEE Trans. on Computer-Aided Design*, 6(6):942–955, Nov 1987.

[47] H. Chen, C. Qiao, F. Zhou, and C. Cheng. Refined single trunk tree: A rectilinear Steiner tree generator for interconnect prediction. In *Proc. ACM Intl. Workshop on System Level Interconnect Prediction*, pages 85–89, 2002.

[48] M. Halldorsson. Approximations of weighted independent set and hereditary subset problems. *Journal of Graph Algorithms and Applications*, 1(4):1–16, Apr 2000.

[49] IBM-Place 2.0 benchmark suits. http://er.cs.ucla.edu/benchmarks/ibm-place2/.

[50] Y. Zhang Y. Xu and C. Chu. FastRoute 3.0: A fast and high quality global router based on virtual capacity. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 344–349, 2008.

[51] E. Bozogzadeh R. Kastner and M. Sarrafzadeh. Predictable routing. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 110–113, 2000.

[52] Y. Xu and C. Chu. An auction based pre-processing technique to determine detour in global routing. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 305–311, 2010.

[53] Y. Zhang and C. Chu. RegularRoute: An efficient detailed router with regular routing patterns. Accepted by *ACM/SIGDA Intl. Symp. on Physical Design*, 2011.

ACKNOWLEDGEMENTS

I would use this opportunity to thank those who give me help and support during my Ph.D. study. First and foremost, I would express my highest gratitude to my major advisor Dr. Chris Chu for his continuous support during my Ph.D. study. Without his guidance and inspiration, I could not finish this work. I'm deeply grateful for his support in various aspects including conducting research, writing papers and presenting my work in technical conferences. I would also like to thank my committee members Dr. Randall Geiger, Dr. Akhilesh Tyagi, Dr. Degang Chen and Dr. Yong Guan for their efforts and contributions in my thesis work. I would also like to thank Dr. Min Pan for his collaboration in taking design contests and suggestions in my early career.