國立交通大學資訊學院資訊科學與工程研究所

博士論文

Graduate Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

Doctoral Dissertation

用於擺置繞線流程的可繞度和效能最佳化技術

Optimizing Routability and Performance of Placement and Routing Flow for Nanometer Designs

劉文皓

Wen-Hao Liu

指導教授:李毅郎 博士

Advisor: Yih-Lang Li, Ph.D.

中華民國一百零二年一月

January, 2013

摘要

近年來，隨著奈米技術的進步，晶片中的元件越來越多，同時繞線的難度也越來越高。晶片的可繞性(Routability)成為眾人所觀注的議題。在該議題上，全域繞線(Global Routing)扮演著重要的角色。在電路實體設計(Physical Design)的流程中，全域繞線上承元件擺放(Placement)，下啟細部繞線(Detailed Routing)。一個快速的全域繞線器能提供擁擠度(Congestion)的資訊給擺放器，讓擺放器擺出較容易繞線的布局。此外，若全域繞線器能有效的解決繞線擁擠度的問題，細部繞線的負擔和所需時間可以大幅降低。當全域繞線器和擺放器合作時，全域繞線器如何快速且準確的回報擁擠度資訊給擺放器是一個重要的議題。另一方面，當全域繞線器扮演細部繞線器的指導者角色時，全域繞線結果的品質就顯的格外重要。若細部繞線器能根據一個高品質的全域繞線結果進行細部繞線，將會提升細部繞線結果的品質，並大縮短細部繞線的時間。這篇論文提出了兩個全域繞線器：Grace是個快速的全域繞線器，適合擔任擁擠度預估器；NCTU-GR 2.0能產生高品質的繞線結果來指導細部繞線，其繞線結果較不擁擠且有較短的線長。此外，為了將Grace應用於業界，我們增加了許多功能於Grace中來滿足業界的需求。在擺放和繞線的中間階段，我們結合了擺放器和全域繞線器，提出了一個可繞性優化器(Routability Optimizer)。若給予一個布局結果，該優化器可以重新擺放其中的元件，讓該布局更容易被繞線，進而得到更好的繞線結果。最終，在進行細部繞線前，我們還提出了一個三維繞線改善器，將全域繞線的結果在做進一步的改善，此舉可以更進一步的降低細部繞線器的時間和負擔。

Abstract

Routability has become one of most critical issues to successfully achieve design closure. To address this issue, global routing plays an important role in the placement and routing flow. During the placement stage, a fast global router can serve as a routing congestion estimator to guide that placers improve the routability of placement solutions; however, traditional global routers are too slow to offer quick but accurate congestion estimation. In the routing stage, the duty of a global router is to identify a global routing result to guide downstream detailed routers. The runtime of the detailed router can significantly reduce if the global routing result has well optimized congestion and wirelength.

In this dissertation, two global routing engines are proposed, Grace and NCTU-GR 2.0. Grace is a fast global router to serve as a fast routing congestion estimator, adopts the proposed unilateral monotonic routing and hybrid unilateral monotonic routing to replace time-consuming maze routing in its routing flow, and invokes a congestion-aware bounding box expansion scheme to avoid over-expanding the searching regions to achieve high speedup. Moreover, in order to use Grace in the industrial flow, Grace have been enhanced to tackle the layer directive and scenic constraints for considering the timing issue.

Another proposed global router NCTU-GR 2.0 can generate high-quality global routing results to guide the downstream detailed router. The proposed bounded-length maze routing avoids producing redundant detours to save routing resource; rectilinear Steiner minimum tree aware routing scheme can guide NCTU-GR 2.0 to build a routing tree for each multi-pin net with shorter wirelength; a dynamically adjusted history cost function highlights for NCTU-GR 2.0 which grid edges are critical routing resource that can be more carefully allocated to the nets that really desire. Based on the proposed innovations to carefully utilize routing resource, NCTU-GR 2.0 obtains shorter total wirelength and lower congestion than the other state-of-the-art academic global routers.

In addition, between the placement and routing stages, this dissertation presents an incremental place-and-route tool called Ropt to optimize the routability of a given placement solution. Rather than minimizing HPWL, Ropt directly improves routability by minimizing the routing cost of nets, as the routing cost is defined in terms of global congestion, local congestion and wirelength. In addition to using NCTU-GR 2.0 to evaluate the routability of the placement solutions, this work also uses Wroute to obtain detailed routing results of the optimized placement solutions for the evaluation of real routability.

Finally, the proposed post-3D-global-router called Post3DGR further refines the wirelength, congestion, and via count of a given 3D global routing result. Post3DGR consists of the 3D post routing stage and negotiation-based layer assignment stage. The 3D post routing stage adopts an inherited history cost function to guide the routing, which can greedily reduce total wirelength and vias. The negotiation-based layer assignment stage re-assigns the routing layer for each wire to reduce via count. The negotiation-based layer assignment can be extended to consider via overflow and antenna effect. Considering these issues before detailed routing can ease the effort and runtime of subsequent detailed routing.

誌謝

能順利的完成這份論文，我最先要感謝的是我的父母–劉修添先生和蘇素華女士。謝謝你們的支持，讓我能無後顧之憂的做自己喜歡的研究。你們的身教和言教，是我能順利完成博士學位的最大原因。謝謝你們。我希望將這份論文獻給我最親愛的父親和母親。

在我博士生涯中，謝謝李毅郎老師的細心且耐心的指導。我何等幸運，因為經師多有，人師難得。謝謝你教導我做研究的方法和待人處事的道理。這四年在你的教導下，我覺得我在很多方面都有大幅的進步，特別是論文撰寫方面。四年前，我連一句文法正確的英文句子都寫不出現。往往整篇論文都要經過你的大幅翻修，才能讓人讀懂。謝謝你花許多精力批改我的論文和教導我如何寫論文。因為你的教導，這篇博士論文才能順利完成。

謝謝Cheng-Kok Koh教授給我到普渡大學進行千里馬的機會。我在普渡大學的這段時間，謝謝你關心我的生活，並跟我分享你的人生經驗。在我們討論研究進度時，謝謝你能忍受我的破爛英文，並且不時的幫我正音。我從你身上學到了有別以往的研究方法，並且更開擴了我的國際觀。我相當感激並且慶幸能受到你的指導。當我在美國的期間，謝謝Cliff Sze博士給我到IBM Austin Lab進行短期研究的機會。IBM Austin Lab一直是我心中所響往的一流研究機構，在這裡，我很慶幸能和許多知名的研究者共事。我在IBM的期間，學到了業界解決問題的方法，並且讓我的英文的溝通能力大有進步。

一路走來，我要感謝的人實在是太多了。謝謝國小時的林蘭春老師，國中時的張澄仁老師，高中時的徐英珠老師和黃玉慧老師。你們總是在

我做錯事時包容我、糾正我;在我意志消沉時鼓勵我、鞭策我。沒有你們的教誨，就沒有今日的我。在我就讀博士期間，謝謝王廷基教授和張耀文教授對我的照顧和經驗分享。我要特別感謝王廷基教授，我在研究上遇到困難時，王廷基教授經常對我伸出援手。我還要感謝我的姑姑-劉虹君女士，謝謝妳對我的照顧和愛護，妳在我的成長過程中扮演著十分重要的角色。最後我要感謝我的女朋友-林珈竹，謝謝妳陪伴我走過研究所的求學生涯，謝謝妳在我研究受挫時鼓勵我、聽我訴苦。有妳相伴的時光，讓我的生活不再只是程式碼和論文，妳讓我的研究所生涯多了許多美好的回憶。謝謝妳，我愛妳。

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Overveiw of this Dissertation

With ceaseless advances in semiconductor technology shrinkage, the main contributing factors to the increasingly more challenging routing problem include the high number of metal layers, wide range of metal thickness, and complex design rules, thus routability has become a critical issue in VLSI physical design flow. To address the routability issue, global routing plays an important role since global routing bridges the gap between placement and detailed routing. In traditional physical design flow, the global routing stage follows the placement stage to yield a rough routing result for most nets, and then the detailed routing stage based on the rough routing result completes physical routes for every net and finally realizes a detailed routing result. For example, Figs. 1.1(a), (b) and (c) respectively show a placement solution, a global routing result and a detailed routing result for a design, in which the gray rectangles denote macros, the small rectangles denote cells, and red circles denote pins of a net. Figure 1.1 illustrates that global routing identifies a set of routing regions that the net should pass through, and detailed routing finds a physical routing path in these regions.

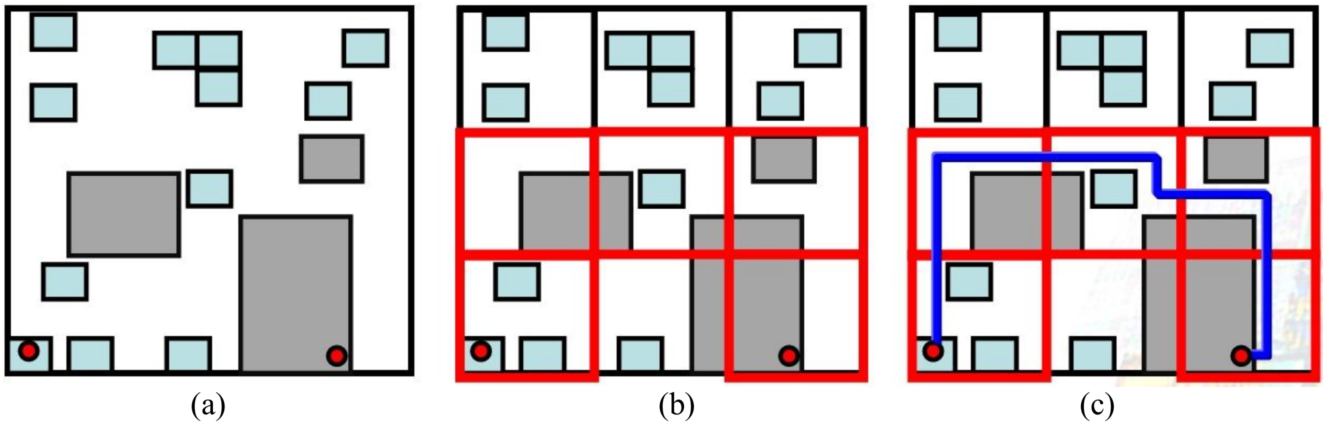


Fig. 1.1 (a) A placement solution; (b) a global routing result; (c) a detailed routing result of a design.

In this dissertation, a routability-driven placement and routing (P&R) flow (Fig. 1.2) is presented based on the proposed global routing engines and post-placement routability optimizer. The proposed global routing engines can not only cooperate with placers to obtain better routability placement solutions, but can also yield high-quality global routing results so that detailed routers can apply the global routing results to generate good detailed routing results. Compared to the traditional placement and routing flow, the flow shown in Fig. 1.2 pays more attention to the interaction between placement and routing and optimizes the main factors to influence routability such like global congestion, local congestion, wirelength and via count, which can detect and fix the congestion problems in the early stages and thus contributes to faster design closure. The red boxes in Fig. 1.2 highlight the contributions of this dissertation to deal with routability issue, introduced in the following four paragraphs.

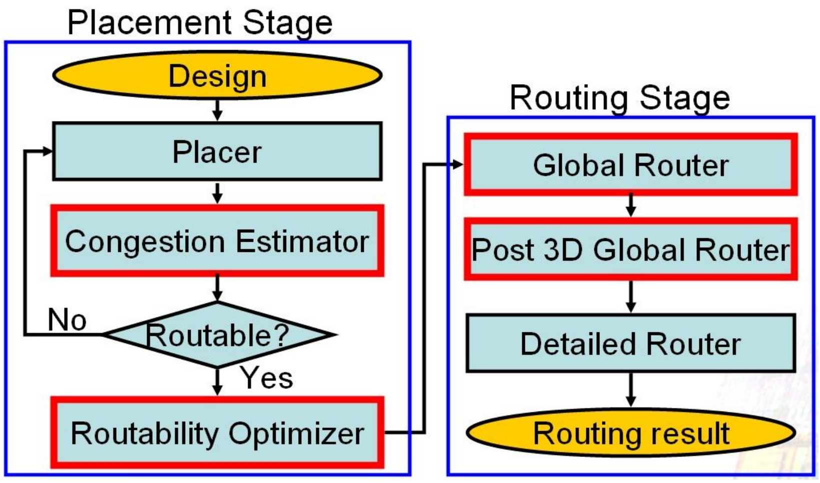


Fig. 1.2 Routability-driven placement and routing (P&R) flow.

To avoid wasting time on routing unroutable designs, a routing congestion estimator (RCE) can help designers to fast judge whether a design is routable in the early stages to speed up the design closure. Also, a RCE can cooperate with placers to optimize routability. Thus, this dissertation presents a global-routing-based RCE cooperating with placers to improve routability. The proposed global-routing-based RCE can offer more accurate routing congestion estimation than probabilistic-based RCE [35, 36] since global-routing-based RCE can better capture the actual routing behaviors. However, global-routing-based RCE is typically slower than probabilistic-based RCEs. Because a RCE may be frequently launched in the placement stage, the placement stage would slow down if the launched RCE is not fast enough. Accordingly, the objective of a global-routing-based RCE is to identify an accurate congestion map as fast as possible. This dissertation presents an accurate and fast **g**lobal-**r**outing-b**a**sed R**CE** called **Grace**. In addition to testing Grace on academic benchmarks, we also enhance Grace to fulfill industrial requirements and then apply the enhanced Grace in the industrial design flow.

In the flow of Fig.1.2, RCE reports the congestion information of a placement solution to placers, and then placers can move cells based on the congestion map to improve the placement solution's routability. However, as cells move, the congestion map also changes, thereby degrading the effectiveness to improve the routability of a placement. To resolve this problem, we develop a routability optimizer **Ropt** that takes a placement solution and optimizes its routability by incremental place-and-route. Ropt always maintains a global routing instance based on the current placement solution. The global routing instance is built on a local-routability-aware model. Therefore, the global routing instance provides both global and local congestion information to guide the placement algorithms. Also, the placement algorithms in Ropt invoke a global routing engine to decide the placed locations for movable cells.

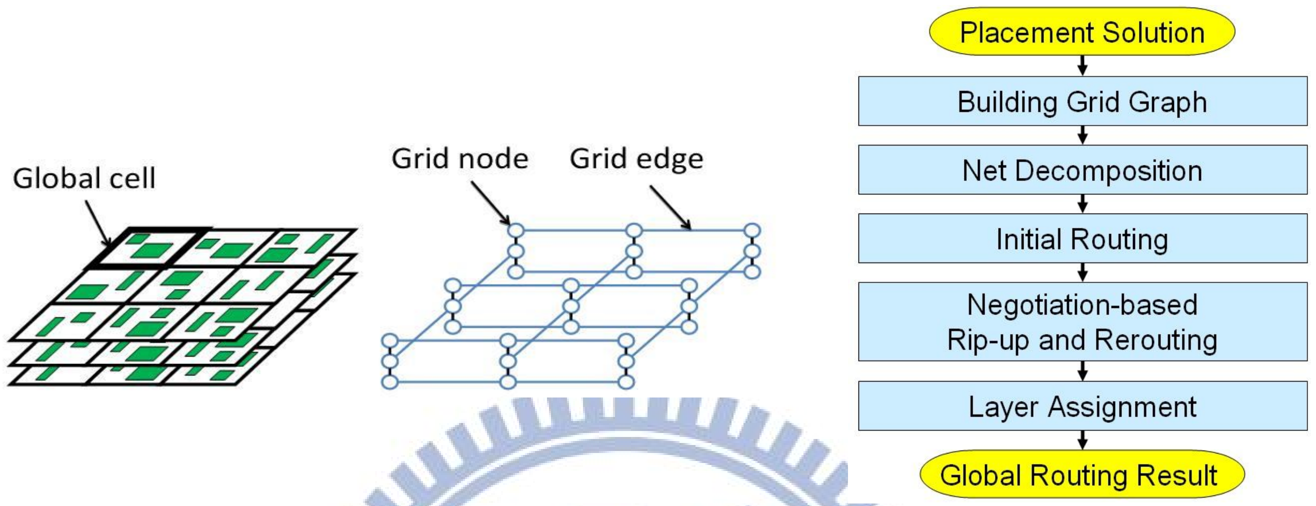
After a placement solution is optimized by Ropt, the proposed global router **NCTU-GR 2.0** is used to identify a high-quality global routing result of the placement solution to provide a good blueprint for detailed routing. Generally, the runtime of the detailed routing stage is hundred or thousand times of that of the global routing stage. Good global routing results can diminish the time of detailed routing and promote the final interconnection quality significantly. Because the routing quality of a global routing result can be measured by the total overflow and total wirelength, minimizing overflows and wirelength is the major task for global routing researches [3-21], where overflow means that a region's routing demand exceeds its routing capacity. Compared to other state-of-the-art global routers [6, 11, 16, 17], the proposed NCTU-GR 2.0 can get global routing results with fewer overflows and shorter wirelength. Note that, although Grace can be treated as a light global router, the algorithms used in Grace and NCTU-GR 2.0 are largely different since their purposes are different. Chapter 5 will detail the algorithmic differences between designing a global-routing-based RCE and a global router.

With semiconductor technology shrinkage, the number of metal layers ceaselessly increases. Thus, the problem of planning routing wires on which metal layers becomes more challenging. The layer planning for routing wires impacts the amount of vias, timing, and many manufacturing issues such like antenna effect and double-vias. However, the typical global routing stage ignores these issues and leaves these issues to detailed routing, which may make detailed routers struggle for these issues. Accordingly, we develop a post-3D-global-routing tool **Post3DGR** between global routing and detailed routing to refine a given 3D global routing result, which can ease the effort of detailed routers to speed up the design closure. The proposed Post3DGR can reduce the vias, congestion, and wirelength of a given 3D global routing result by re-routing nets and re-planning wires' layers. With some modifications, Post3DGR also can take antenna effect into account.

The rest of this dissertation is organized as follows. Chapter 1 introduces the problem formulation and background of global routing. Chapter 2 presents a fast global-routing-based RCE called Grace whose goal is to identify a satisfactory global routing result to predict routing congestion as fast as possible. Chapter 3 presents an enhanced Grace applied in the industrial flow to consider timing and local congestion, and target congestion ratio. Chapters 4 introduces the proposed incremental place-and-route tool Ropt that can optimize the routability of a given placement solution. Chapter 5 presents a global router called NCTU-GR 2.0 whose objective is to obtain high-quality global routing results in a reasonable runtime to guide detailed routers. A post-3D-global-routing tool Post3DGR is detailed in Chapter 5. Finally, Chapter 7 draws conclusions.

1.2 Background

In global routing problem, typically the given placement solution is partitioned into a 3-dimension (3D) array of global cells (Fig. 1.3(a)), and then the array of global cells is modeled to a 3D grid graph (Fig. 1.3(b)). Generally, there are two strategies to deal with global routing problem on the 3D grid graph. One directly performs global routing on a 3D grid graph [3-6]. Although directly performing global routing on a 3D grid graph may achieve a better result, it is time-consuming. Thus, the mainstream approach is to condense 3D grid graph into 2D grid graph first, and then peform 2D global routing to obtain a 2D routing result. Finally, layer assignment algorithms [17, 27-33] assign each routing wire to the corresponding metal layers to obtain a 3D global routing result [7-21]. Figure 1.3(c) shows the general flow adopted in most global routers to tackle 3D global routing porblem, the functions of each stage are detailed in the follows.



(a) (b) (c)

Fig. 1.3 (a) partition a placement into a 3D array of G-cells; (b) model the 3D array of G-cells into a grid graph; (c) typical global routing flow.

1.2.1 Building Grid Graph and the Objectives of Global Routing

In the grid graph, each grid node refers to a global cell (G-cell), and each grid edge corresponds to a boundary between two abutting global cells in the same layer. Meanwhile, each via edge connects two abutting G-cells in two adjacent layers. The number of routing tracks that can be accommodated across the abutting boundary is defined as the capacity $c(e)$ of a grid edge e, and the number of wires that pass through grid edge e is called grid edge's demand $d(e)$. The overflow of a grid edge e is defined $\max(d(e)-c(e), 0)$, the total overflow is the sum of overflows on all grid edges, and the maximum overflow is the maximum overflow among all edges. For simplicity, the capacity of each via edge is not limited, which is also adopted in most of global routing researches [3-21]. Given the pins' locations of each net distributed on the grid graph, the objective of global routing problem is to identify a highly routable global path to connect the pins of each net. The quality of a global routing result is generally measured by the total overflow and wirelength.

Figure 1.4 shows how to compute the capacity of 2D grid edges in the mainstream flow of 2D global routing with layer assignment, in which the numbers next to 3D grid edges denote the capacity of the 3D edges. After the 3D grid graph is compacted to a 2D graph, the capacity of a 2D grid edge is obtained by adding up the capacities of its corresponding 3D grid edges.

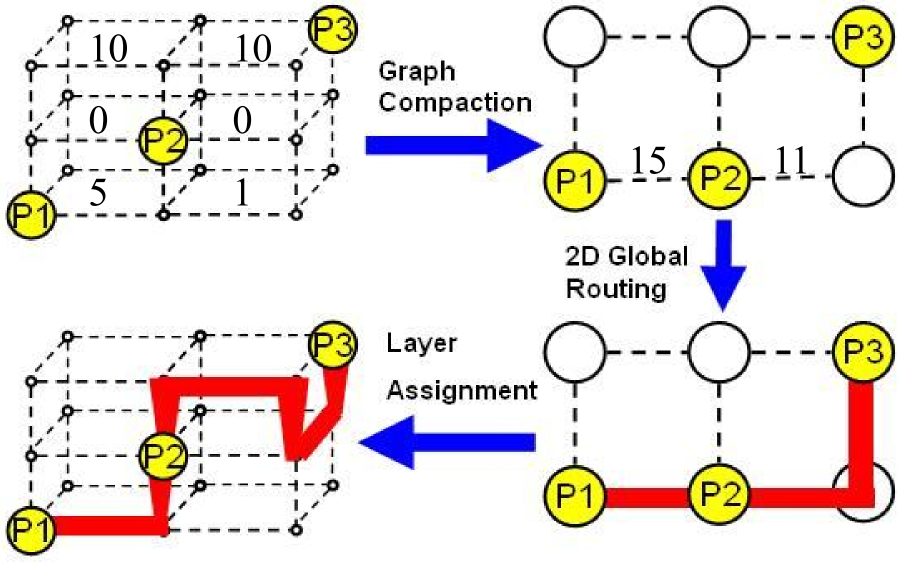


Fig. 1.4. modern 3D global routing flow.

1.2.2 Net Decomposition

Most global routers decompose each multi-pin net into two-pin subnets, because net decomposition can simplify a multi-terminal routing problem to a two-terminal routing problem. Before routing stages, the rectilinear Steiner minimal tree (RSMT) or rectilinear minimum spanning tree (RMST) construction algorithms are commonly used to generate the initial topology for each multi-pin net and then each multi-pin net is decomposed into two-pin subnets based on its topology. For example, Figs. 1.5(a) and 1.5(b) show the initial topologies of a four-pin net generated by RSMT and RMST, respectively, in which the green rectangle denotes a Steiner point, and the topologies of the four-pin net in Figs. 1.5(a) and 1.5(b) can be decomposed to 4 and 3 two-pin subnets, respectively. Because a RSMT has shorter wire length than a RMST has, net decomposition by RSMT is popular in many literature. FLUTE [23] is a very fast and accurate RSMT construction tool, which is widely used by many modern global routers. FLUTE not only quickly constructs a good RSMT for a multi-pin net, but also obtains optimal RSMTs for nets with nine or fewer pins. However, FGR [3] indicates that the RSMT has less routing flexibility than the RMST as it owns Steiner points and generates more flat segments than the RMST, and the used data structure of RSMTs is more complex than that of RMSTs. On the contrary, the RMST can simply complete each subnet's routing with pattern routing or monotonic routing to avoid congestion regions. Consider wirelength and routing flexibility, in which a RMST that encourages multiple two-pin routings to merge together with multiple paths that pass through the same grid edges (Fig. 1.5(c)). This ideal solution avoids passing through congested regions by using a shorter total wire length than that of a RMST that does not encourage finding joint wires. However, how to identify a RMST with joint wires is a challenge.

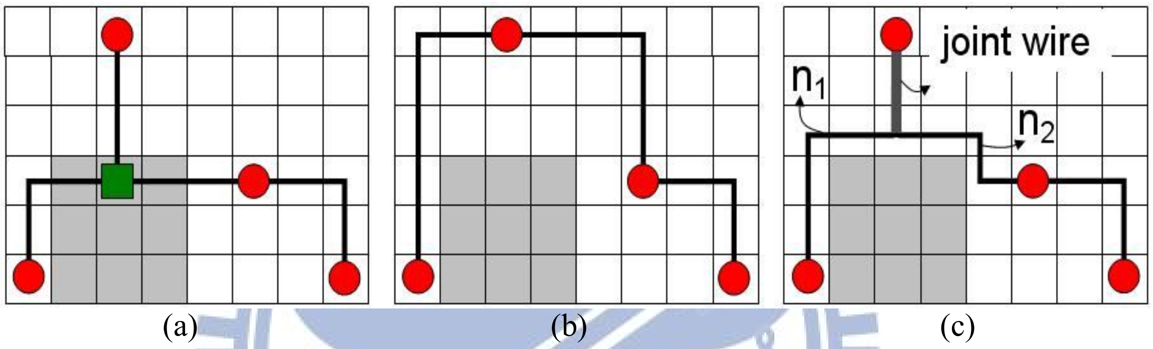


Fig. 1.5. Four-pin net decomposition by (a) RSMT; (b) RMST; (c) RMST with a joint wire, subnets n_1 and n_2 share a joint wire.

1.2.3 Pattern Routing and Monotonic Routing

Pattern routing adopts specific routing patterns to connect two pins. The most common patterns are L-shaped or Z-shaped. The main advantage of pattern routing is that it can complete the path searching in a very short time, but its solution space is very tiny. To mitigate the huge performance gap between pattern routing and maze routing, Pan *et al.* [14] present monotonic routing to enrich the solution space. Monotonic routing uses the dynamic-programming technique to identify a routing path from the source to the target without any detour. The time complexity of monotonic routing in a $m \times n$ grid graph is $O(mn)$, which is the same as that of the Z-shaped pattern routing.

1.2.4 Negotiation-based Rip-up and Rerouting (NRR)

Rip-up and re-routing technique is widely used in global and detailed routing. Given an illegal routing solution, rip-up and rerouting technique iteratively removes the nets with violations and reroutes them sequentially to expel violations. In global routing problem, a violation occurs when an overflow is produced. Widely, the negotiation technique, as proposed in PathFinder [22], is associated with rip-up and re-routing technique (NRR) in modern global routers to improve the ability of overflow removal. The main idea of NRR is to increase the penalty of a grid edge at current iteration that overflowed at the previous iteration. Thus, path searching intends to avoid passing previously overflowed grid edges. [22] formulates the negotiation-based routing cost of grid edges *e* as follows,

$$c_e = (b_e + h_e) \times p_e \, , \tag{1.1}$$

where c_e represents the routing cost of *e*; b_e denotes the base cost; h_e denotes the history cost, and p_e denotes the congestion penalty. The history cost h_e increases as overflow occurs. The value of h_e in the (*k*+1)-th iteration is given by:

$$h_e^{k+1} = \begin{cases} h_e^k + h_{inc} & \text{if } e \text{ is overflowed} \\ h_e^k & \text{otherwise} \end{cases} , \tag{1.2}$$

where $h_e^1 = 1$, h_{inc} is a constant, and h_e^k is updated in every iteration. In addition, FGR [3] presents another formula to preserve the base cost as follows.

$$c_e = b_e + h_e \times p_e \, . \tag{1.3}$$

Several variations of negotiation-based cost functions have been discussed in [10-12, 16-17].

1.2.5 Layer Assignment

The goal of layer assignment in global routing is to translate a 2D global routing result into a 3D result on minimizing the number of vias while not changing routing topology or increasing any overflows, which is called the congestion-constraint layer assignment problem. Congestion-constrained layer assignment problem for via minimization has been proven to be NP-complete [34] and extensively studied. BoxRouter2.0 [9] adopted integer linear programming to minimize via count minimization. FGR [3] greedily assigned net edges to the corresponding metal layers by heuristics. Lee *et al*. proposed an efficient sequence layer assignment algorithm called COLA [27], which determined net assignment order at first and then assigning each net to the appropriate layer by a dynamic-programming technique. FastRoute 4.0 [16] decomposes multi-pin nets to two-pin net, then using the dynamic-programming algorithm to assign each two-pin net one bye one. Dai *et al.* [17] presented a congestion-relaxed layer assignment with a layer shifting algorithm, followed by net rip-up and re-assigning to further reduce the number of vias. In addition, some researchers extended the layer assignment problem to consider via overflow [28-29], double patterning [30], timing [31], and antenna effect [32-33].

1.2.6 Comparison of Recent Global Routers

Table 1.1 lists the well-know global routers developed in recent six years. Although most global routers in Table 1.1 are based on the global routing flow shown in Fig. 1.3(c), they have different opinions on several issues. Table 1.2 shows the issues that are widely discussed in recent global routing researches. For instance, the routers in [3, 7, 71] use RMST to be the initial tree topology for each net, while the routers in [16, 71] use RSMT; NTHU-Route [11] reroutes the nets in the un-congested region earlier, while the routers in [3, 12, 17] reroutes the nets in the congested region earlier; Box-Router [9, 70] rips-up a set of nets and then reroutes these nets one by one, while the routers in [4, 7, 11, 17] rip-up a net and then reroute it immediately. On the parallel routing issues, GRIP [4, 5] parallelize global routing on a cluster computing environment, NCTU-GR [18, 71] performs on a many-core server, the router in [19] performs on a GPU-CPU hybrid platform.

TABLE 1.1 RECENT GLOBAL ROUTING RESEARCHES

NTHU-Route [69, 11]	FastRoute [13-16]	FGR [3, 7]	MGR [6]
NTUgr [12]	Box-Router 2.0 [70, 9]	NCTU-GR [17, 18, 71]	Archer [10]
GRIP [4, 5]	HybridGR [19]	Maize-Router [8]	

TABLE 1.2 THE ISSUES DISCCUSED IN RECENT GLOBAL ROUTING RESEARCHES

Net decomposition	[3, 7 11, 16, 71]	Routing algorithms	[6, 8, 10, 12, 14, 16, 71]
Routing nets ordering	[3, 11, 12, 17]	Layer assignment approaches	[3, 9, 16, 17]
Rip-up and rerouting scheme	[4, 7, 9. 11, 17, 70]	Routing cost formulation	[3, 7, 10, 11, 12, 13, 16, 17, 69, 71]
Multi-threaded routing	[4, 5, 19, 71]		

Chapter 2 Grace: A Fast Global-routng-based Routing Congestion Estimator

2.1 Introduction

Routability is of primary concern in nanometer-scale design. Considering the routability issue in placement stage can avoid generating an unroutable design. Two strategies are generally adopted by routability-driven placement to estimate the congested regions (hot spots). First, the probabilistic method estimates the routing congestion of a region by using the pin density and the nets' bounding box or Steiner tree [35, 36]. Although fast, this method typically fails to capture actual routing behavior, and therefore has low estimation accuracy. The second congestion estimation strategy performs global routing to analyze routing congestion [37]. The latter method can identify more precisely the congestion information. However, such an approach is markedly slower than the former one. Among the modern routability-driven placers, Ripple [38], NTUplace [39] and the placers in [40, 41] used the former strategy, whereas SimPLR [42], IPR [43], CRISP [44] and GRplacer [45] adopted the latter one. Clearly, it is inevitable to trade-off routing quality for better run-time performance when these built-in global routers are concerned.

Maze routing with A* search scheme is the indispensable kernel algorithm of state-of-the-art global routers [3-19]. For hard-to-route benchmarks, these routers attempt to eliminate overflows by iteratively ripping up and rerouting overflowed nets by using maze routing. However, maze routing is slower than other routing algorithms, such as pattern routing and monotonic routing algorithms. Several works have attempted to reduce runtime by developing alternative routing algorithms in order to lower the frequency of invoking maze routing. For instance, Archer [10] developed the U-shaped pattern routing algorithm; NTUgr [12] presented the escaping routing algorithm; and FastRoute 4.0 [16] developed the 3-bend routing algorithm. These routing algorithms run faster than maze routing within a quite limited solution space. Thus, in these global routers, maze routing continues to be the last-gasp approach to identify better routes. Consequently, maze routing still consumes the majority of the runtime in the entire routing flow.

This work presents an extremely fast global router called **Grace**, which does not include maze routing to achieve high speedup as an ideal built-in routing congestion estimator for placers.

(a) This work presents two efficient routing algorithms, called unilateral monotonic routing and hybrid unilateral monotonic (HUM) routing. HUM routing can identify a better routing path than U-shaped pattern routing, 3-bend routing, and escaping routing. Moreover, the time complexity of HUM routing is the same as those of these three approaches, linear in terms of the size of the routing region.

(b) Many routers adopt bounding boxes to limit the searching region of routing. Consequently, the bounding box size affects the routing quality and runtime. This work presents an efficient congestion-aware bounding box expansion scheme. With this scheme, the proposed router can improve runtime by 50% than without this scheme.

(c) The proposed router relies on HUM routing to eliminate overflows without invoking maze routing. Experimental results indicate that the proposed router achieves a routing quality similar to that of the proposed maze-routing-based router NCTU-GR 2.0 [18]. Moreover, the run-times of the proposed router are up to 26 times faster than those of [18] on large benchmarks.

The rest of this chapter is organized as follows. Section 2 introduces the global routing problem and the research objective. Section 3 then presents the proposed unilateral monotonic routing, HUM routing algorithms and a congestion-aware bounding box expansion scheme. Section 4 displays the design flow of the proposed global router. Section 5 summarizes the experimental results. Conclusions are finally drawn in Section 6.

2.2 Problem Description

Global routing is formulated as the routing problem on a grid graph $G(V, E)$, where V denotes the set of grid cells, and E refers to the set of grid edges. Each grid edge is termed by the proximity of the related G-cells to its two end nodes. The capacity $c(e)$ of a grid edge e indicates the number of routing tracks that can legally cross the abutting boundary. The number of wires that pass through grid edge e is called the demand of the grid edge $d(e)$. The overflow of a grid edge e is defined as follows. The total overflow is the sum of overflows on all edges of E.

$$overflow(e) = \begin{cases} (d(e) - c(e)) * (w_L + s_L), if\ d(e) > c(e) \\ 0 \qquad ,otherwise \end{cases} \qquad (2.1)$$

where w_L and s_L respectively denote the minimum wire width and wire spacing at layer L where e belongs. In modern designs, higher layers have larger wire width and wire spacing.

Conventionally, overflow and wirelength minimizations have a higher priority than runtime improvement for global routing that offers a global path to guide the detailed routing of each net. However, when global router plays the role as a congestion estimator, the runtime issue become more critical because the estimator have to report the congestion information to placers in a limited time budget (e.g. around 1~5 min). Accordingly, this work focuses on comply with the limited time budget to complete global routing.

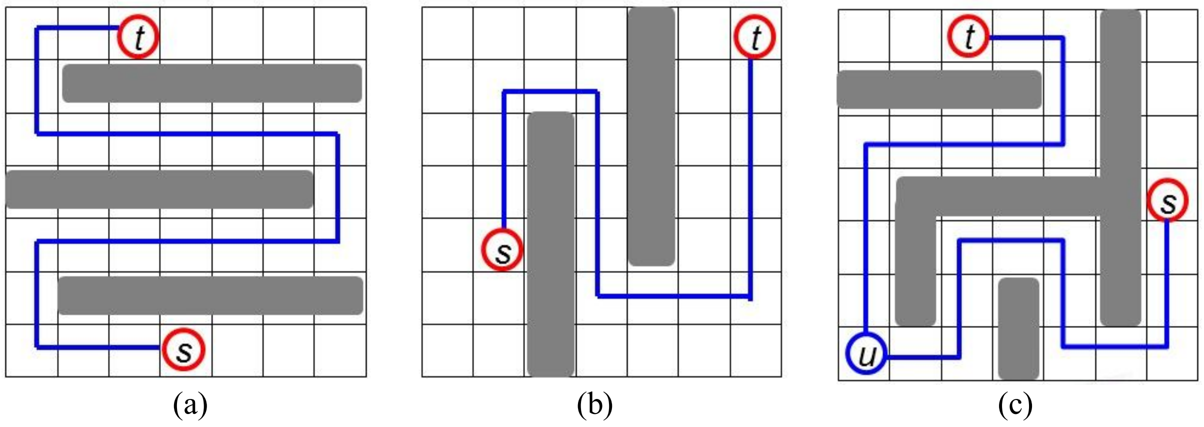


Fig. 2.1. (a) Vertically monotonic routing path; (b) horizontally monotonic routing path; (c) routing path combining vertically and horizontally monotonic routing.

2.3 The Proposed Algorithms for Accelerating Routing

Although capable of identifying a detour-free path efficiently, monotonic routing fails to replace maze routing when a detoured path is required to avoid obstacles or congested regions. A detour is viewed as a move away from the target. To approach the behavior of maze routing, we develop an extremely fast routing algorithm, called *unilateral monotonic routing*, capable of seeking a detoured path and running in the same time complexity as that of monotonic routing. Unilateral monotonic routing identifies a least-cost routing path within a limited region using minimal horizontal or vertical distance. Two unilateral monotonic routing types are defined as follows.

Definition. *Horizontally/Vertically monotonic* (*HM/VM*) *routing* identifies the least-cost routing path from the source to the target using minimal horizontal/vertical distance.

For a HM/VM routing path, a detour occurs only in vertical/horizontal move. Figures 2.1(a) and 2.1(b) illustrate a VM routing path and a HM routing path, respectively, in which the gray rectangles represent congested regions. Although the solution space of HM or VM routing is less than that of maze routing, alternatively invoking HM and VM routings together can increase the solution space significantly. Figure 2.1(c) depicts an example of invoking successive HM and VM routings, the path in Fig. 2.1(c) consists of a HM routing path from *s* to an internal node *u* and a VM routing path from *t* to *u*.

2.3.1 Unilateral Monotonic Routing

Without a loss of generality, the proposed unilateral monotonic routing is introduced by using an example of VM routing shown in Fig. 2.2. At the beginning of VM routing, the congestion map (Fig. 2.2(a)) is formulated into the global routing model (Fig. 2.2(b)), and the congestions is formulated into the routing cost on each grid edge, then a window is given to enclose the source and target with the height of vertical distance between the source and target and the width of horizontal distance larger than that between the source and target (Fig. 2.2(b)). The window size determines the runtime and the routing quality of the unilateral monotonic routing. Section 3.3 in this chapter will introduce how to determine the window size. Figure 2.3 shows the pseudo code of the VM routing algorithm, in which source *s* and target *t* are located at (x_1, y_1) and (x_2, y_2) respectively; *B.l* and *B.r* represent the left and right borders of windows *B,* respectively; $cost(v, u)$ denotes the routing cost of grid edge $e(v, u)$; $d(u)$ refers to the least cost of the VM routing path within *B* from *s* to *u*; and $\pi(u)$ is the predecessor of *u*. The algorithm in Fig. 2.3 consists of two stages. The first stage calculates the $d(u)$ value of each node of the

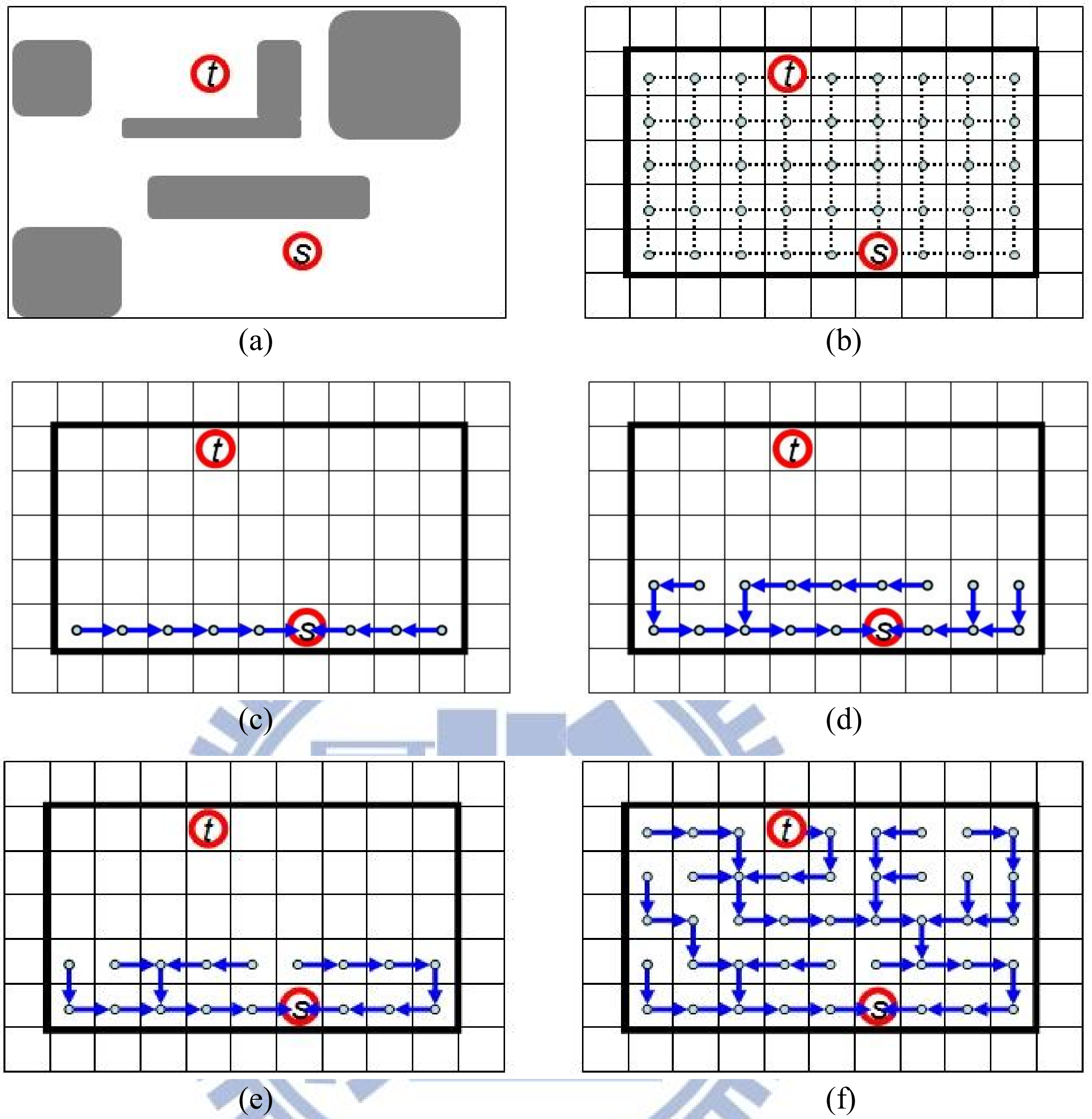


Fig. 2.2. Example of VM routing. (a) a congestion map; (b) the routing model of (a), the dotted lines denote the grid edges; (c) The predecessor of each node *u* in the row of *y*-coordinate y_l after $d(u)$ is obtained, the arrow of each node denotes its predecessor; (d) the predecessor of each node *u* in the row of *y*-coordinate y_l+1 after $lc_{lb}(u)$ is obtained; (e) the predecessor of each node *u* in the row of *y*-coordinate y_l+1 after $d(u)$ is obtained; (f) the routing result of VM routing.

bottom row, i.e. the row where the start node belongs. The second stage computes the $d(u)$ values of nodes in all rows, except for the bottom row, from the row above the bottom row to the top one. The first stage is a simple sequential examination initiating from the start node towards the left and right boundaries of *B*, and then the second stage processes all rows except for the bottom one sequentially and upwards. In the second stage, based on the dynamic programming algorithm, a two-phase flow is developed and the $d(u)$ value of each node is computed row by row. The first phase determines the least-cost VM path to connect every node from the start node at the left or bottom side, while the second phase determines the least-cost VM path to connect every node from the start node at the right side. By the two-phase operation, the least-cost VM path to reach every node within *B* from the start node is then identified.

```
Algorithm Vertically monotonic routing
Input: source $s(x_1, y_1)$, target $t(x_2, y_2)$, bounding box $B$, cost array $d$
1.  $d(s)= 0, \pi(s)= null;$
2.  for $x= x_1-1$ to $B.l$
3.      $u=(x, y_1), v=(x+1, y_1);$
4.      $d(u)= d(v)+ cost(v, u), \pi(u)= v;$
5.  end for
6.  for $x= x_1+1$ to $B.r$
7.      $u=(x, y_1), v=(x-1, y_1);$
8.      $d(u)= d(v)+ cost(v, u), \pi(u)= v;$
9.  end for
10. for $y= y_1+1$ to $y_2$
11.     $u=( B.l , y), v=( B.l, y-1)$
12.     $lc_{lb}(u)= d(v)+ cost(v, u), \pi(u)= v$
13.     for $x= B.l +1$ to $B.r$
14.         $u=(x, y), v_1=(x-1, y), v_2=(x, y-1);$
15.         if $lc_{lb}(v_1) + cost(v_1, u) < d(v_2) + cost(v_2, u)$
16.             $lc_{lb}(u)= lc_{lb}(v_1)+ cost(v_1, u), \pi(u)= v_1$
17.         else
18.             $lc_{lb}(u)= d(v_2)+ cost(v_2, u), \pi(u)= v_2$
19.     end for
20.     $u=( B.r , y), d(u) = lc_{lb}(u)$
21.     for $x= B.r -1$ to $B.l$
22.         $u=(x, y), v_3=(x+1, y), d(u) = lc_{lb}(u)$
23.         if $d(v_3) + cost(v_3, u) < d(u)$
24.             $d(u) = d(v_3) + cost(v_3, u), \pi(u)= v_3$
25.     end for
26. end for
```

Fig. 2.3. The proposed vertically monotonic routing algorithm.

Upon commencement of the second stage, the $d(v)$ value of each node $v \in (i, y_1)$ for $B.l \leq i \leq B.r$ is identified. By assuming that node *u* is located at (i, y_1+1), $lc_{lb}(u)$ is the least of all costs of the VM routing paths from s to u when the predecessor of u is at its left or bottom side, and can be obtained via the following equation in the first phase,

$$lc_{lb}(u) = \begin{cases} d(v_2) + cost(v_2, u), & \text{if } u \text{ is on the left boundary of } B \\ \min(lc_{lb}(v_1) + cost(v_1, u), d(v_2) + cost(v_2, u)), & \text{otherwise} \end{cases} \tag{2.2}$$

where v_1 and v_2 represent the left and bottom adjacent nodes of u, respectively. During the second phase, the least cost of VM paths to reach node u from the start node at the right side (denoted by $lc_r(u)$) and then the least-cost VM path to reach node u from the start node are determined sequentially by the following equation.

$$d(u) = \begin{cases} lc_{lb}(u), & \text{if } u \text{ is on the right boundary of } B \\ \min(lc_{lb}(u), lc_r(u) = d(v_3) + cost(v_3, u)), & \text{otherwise} \end{cases} \tag{2.3}$$

where v_3 represents the right adjacent node of u. If u is on the right boundary of B, the predecessor of u must be on the left side or on the bottom side of u; thus $d(u)$ equals $lc_{lb}(u)$. While each node u is examined sequentially from right to left in the second phase, the least-cost VM path to reach each node from the start node is then determined by Eq. (2.3).

In Fig. 2.3, lines 1 to 9 calculate the least cost of the VM paths from s to each node $v \in (i, y_1)$ for $B.l \leq i \leq B.r$ (Fig. 2.2(c)). Next, based on the dynamic programming method, the least-cost VM path from s to each node of each row within B is identified from the row of y-coordinate y_1+1 to the row of y-coordinate y_2, (lines 10 to 26), where lines 11 to 19 identify the values of $lc_{lb}(u)$ by Eq. (2.2) and lines 20 to 25 identify the values of $d(u)$ by Eq. (2.3). Figure 2.2(d) shows the predecessor of each node u in the s-to-u path of $lc_{lb}(u)$ in the row of y-coordinate y_1+1. Meanwhile, Fig. 2.2(e) shows the predecessor of each node u in the s-to-u path of $d(u)$ in the row of y-coordinate y_1+1. Upon completion of the VM routing, each node within B has a least-cost VM path to reach s along its predecessor (Fig. 2.2(f)). Therefore, the least-cost VM path from s to t is also identified. Obviously, the time complexity of unilateral monotonic routing algorithm is O($|B|$) where $|B|$ represents the area size of B.

```
Algorithm Hybrid Unilateral Monotonic Routing
Input: source node s, target node t, bounding box B
1. Initialize cost array $Ary_{vs}$, $Ary_{vs}$, $Ary_{hs}$, $Ary_{ht}$
2. //Find the paths from each node in B to s
3. Vertically_Monotonic_Routing(s, B.bl, B, $Ary_{vs}$)
4. Vertically_Monotonic_Routing(s, B.tr, B, $Ary_{vs}$)
5. Horizontally_ Monotonic_Routing(s, B.bl, B, $Ary_{hs}$)
6. Horizontally_Monotonic_Routing(s, B.tr, B, $Ary_{hs}$)
7. //Find the paths from each node in B to t
8. Vertically_Monotonic_Routing(t, B.bl, B, $Ary_{vt}$)
9. Vertically_Monotonic_Routing(t, B.tr, B, $Ary_{vt}$)
10. Horizontally_ Monotonic_Routing(t, B.bl, B, $Ary_{ht}$)
11. Horizontally_ Monotonic_Routing(t, B.tr, B, $Ary_{ht}$)
12. foreach node u in B
13.     $mrc(u)=\min(Ary_{hs}(u), Ary_{vs}(u))+\min(Ary_{ht}(u), Ary_{vt}(u))$
14. end foreach
15. Select the node u in B with the least mrc(u), and then trace back from this node to s and t.
```

Fig. 2.4. The pseudo code of hybrid unilateral monotonic routing algorithm.

2.3.2 Hybrid Unilateral Monotonic Routing

Compared to maze routing, unilateral monotonic routing still offers a limited solution space to solve overflows. This section introduces a hybrid unilateral monotonic (HUM) routing algorithm to search for larger solution space than unilateral monotonic routing offers. The HUM routing concept assumes that each node within *B* can be an intermediate point connecting the start and target nodes. The HUM path consists of two paths, i.e. the path linking the start node with an intermediate point and the path linking the intermediate point with the target node. Each path can be formed by unilateral monotonic routing.

Since a path can be formed by VM or HM routing, four combinations are available to form a HUM routing path. By assuming that bounding box *B* encloses nodes *u* and *v*, $VMP_{B(u,v)}$ and $HMP_{B(u,v)}$ represent a VM routing path and a HM routing path connecting *u* with *v* within *B,* respectively. A HUM routing path connecting *s* with *t* belongs to one of the following four path types: ($VMP_{B(s,u)}$, $VMP_{B(u,t)}$), ($VMP_{B(s,u)}$, $HMP_{B(u,t)}$), ($HMP_{B(s,u)}$, $VMP_{B(u,t)}$) and ($HMP_{B(s,u)}$, $HMP_{B(u,t)}$) for each node *u* within *B*. Whereas ($VMP_{B(s,u)}$, $VMP_{B(u,t)}$) denotes a path concatenation operation that combines two unilateral monotonic paths of one or two type to form a HUM path linking start and end nodes. Figure 2.4 shows the proposed HUM routing algorithm. The least costs of $VMP_{B(s,u)}$, $VMP_{B(t,u)}$, $HMP_{B(s,u)}$ and $HMP_{B(t,u)}$ of each node are stored in the arrays Ary_{vs}, Ary_{vt}, Ary_{hs} and Ary_{ht}, respectively, while $B.bl$ and $B.tr$ represent the nodes at the bottom-left and top-right corners of B, respectively. Lines 3 – 6 regard s as the start node, and $B.bl$ and $B.tr$ as pseudo targets. Then, lines 3 and 4 invoking VM routing from s to the pseudo targets obtain VM routing paths from s to every node within B; lines 5 and 6 invoking HM routing from s to the pseudo targets obtain HM routing paths from s to every node within B. Similarly, lines 8 – 11 regard t as the start node and $B.bl$ and $B.tr$ as the pseudo targets, and then obtain VM routing paths and HM routing paths from t to every node within B. Accordingly, lines 2 – 11 identify the value of each element in Ary_{vs}, Ary_{vt}, Ary_{hs} and Ary_{ht}. Thereafter, the least costs of $VMP_{B(s,u)}$, $VMP_{B(t,u)}$, $HMP_{B(s,u)}$ and $HMP_{B(t,u)}$ for each node u within B are obtained (Fig. 2.5(a)-(d)). The algorithm then selects the least-cost HUM routing path among the candidates of four path types (lines 12 – 15).

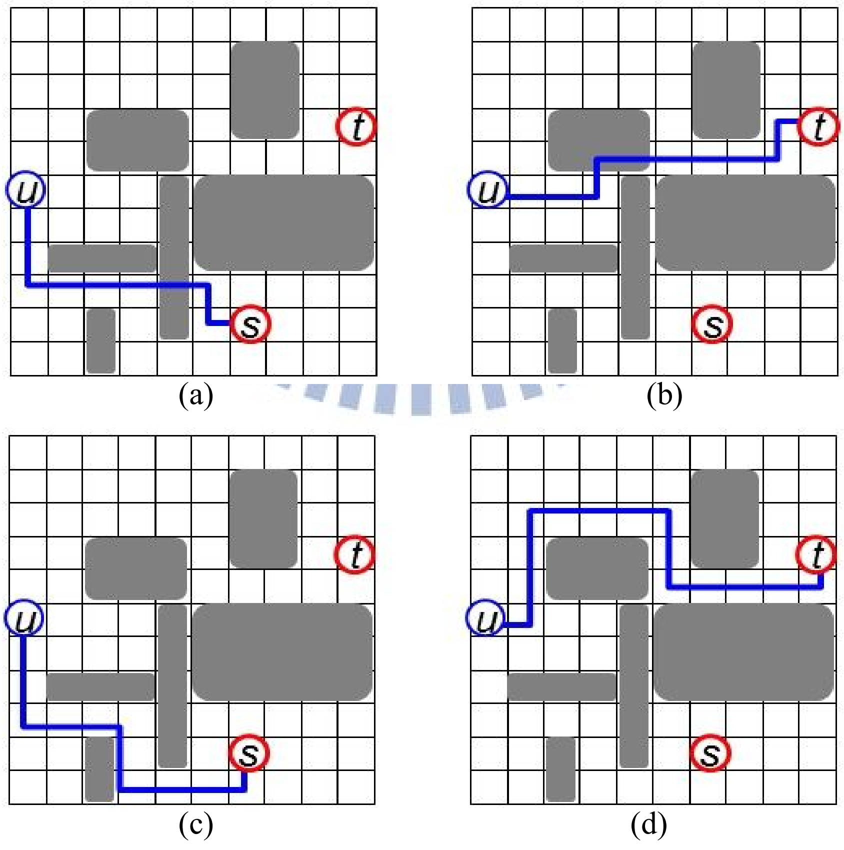


Fig. 2.5. Four path types in B with congested regions (gray rectangles). (a) $VMP_{B(s,u)}$, (b) $VMP_{B(t,u)}$, (c) $HMP_{B(s,u)}$, and (d) $HMP_{B(t,u)}$.

The time complexities of three parts, lines 2-11, lines 12-14 and line 15 are all O(|*B*|). Correspondingly, the time complexity of HUM routing algorithm is still O(|*B*|), which is faster than that of maze routing with A* search scheme (O(|*B*|*log*|*B*|)). Figure 2.6 compares the proposed HUM routing with 3-bend routing and escaping routing, indicating that the time complexities of 3-bend routing and escaping routing are also O(|*B*|). Figures 2.6(a) and 2.6(b) summarize the routing results of 3-bend routing and escaping routing with two overflows and with an overflow, respectively. In contrast, the proposed HUM routing algorithm can identify an overflow-free path (Fig. 2.6(c)) with the pattern ($HMP_{B(s,u)}$, $HMP_{B(u,t)}$). Notably, even if HUM routing cannot identify an overflow-free path, it can always identify a least-cost HUM path which must cost less than or equal to that of 3-bend routing and escaping routing. Because, the solution space of HUM routing totally covers and is much larger than that of 3-bend routing and escaping routing.

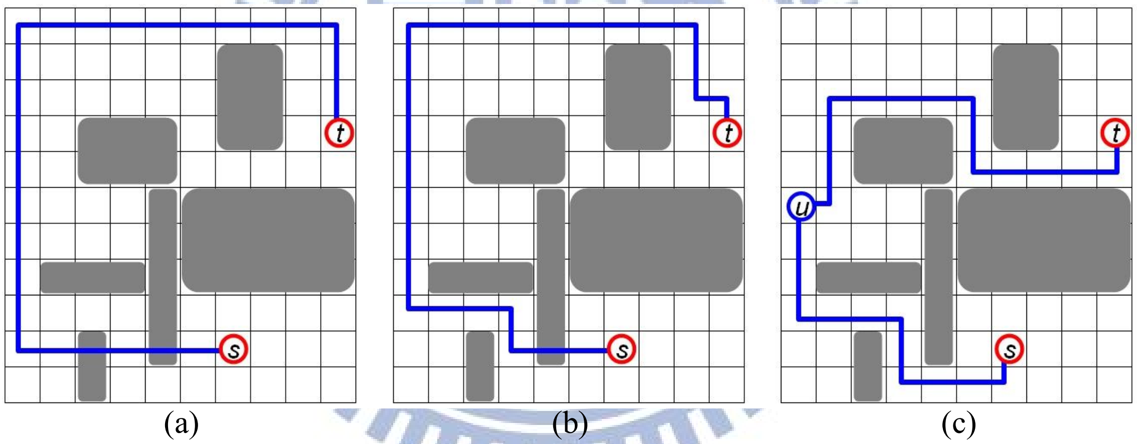


Fig. 2.6. (a) Routing result of 3-bend routing with two overflows; (b) routing result of escaping routing with an overflow; (c) routing result of the proposed HUM routing without overflows.

Assume that most of overflowed grid edges within *B* are aligned in a row similar to the congestion map in Fig. 2.1(a), the least costs of $HMP_{B(s,u)}$ and $HMP_{B(u,t)}$ are likely larger than the least costs of $VMP_{B(s,u)}$ and $VMP_{B(u,t)}$. Therefore, the operations of exploring $HMP_{B(s,u)}$ and $HMP_{B(u,t)}$ can be regarded as redundant and are thus omitted. Based on this observation, four HUM routing types are explored only once for every net at the first time when it is routed by HUM routing. If a net is rerouted by HUM routing in the later routing stage, only the HUM routing type that initially identified the least-cost path is invoked. By this scheme, experimental results indicate that similar routing quality and an approximately 23% decrease in runtime of HUM routing can be achieved.

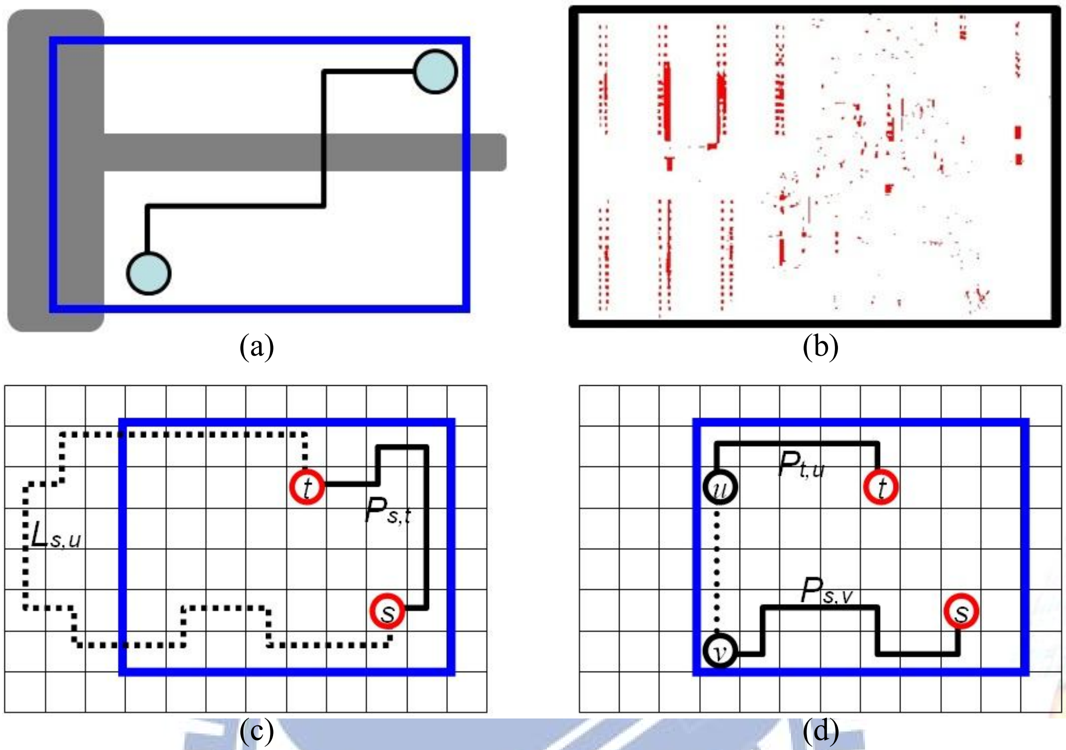


Fig. 2.7. Example of congestion-aware bounding box expansion. (a) Routing path with an vertical overflow; (b) the overflow map of the benchmark superblue1 after the initial routing; (c) currently identified path $P_{s,t}$ and path $L_{s,t}$ that is expected to be across the left side of the bounding box; (d) the estimated lower bound cost of $L_{s,t}$ is the sum of the costs of $P_{s,v}$ and $P_{t,u}$ plus $manh(v, u)* \alpha$.

2.3.3 Congestion-aware Bounding Box Expansion

Bounding box is widely adopted to limit the searching region of routing. In conventional global routers, the initial bounding box is slightly larger than the minimum rectangle enclosing the terminals of the routed net. The inability to identify an overflow-free path within the bounding box causes the bounding box to expand and, then, the overflowed net is rerouted again. The box expansion policy based on current congestion information has seldom been discussed in the literature. The traditional box expansion scheme tends to over-expand, subsequently increasing the runtime. For instance, Fig. 2.7(a) shows a routing path with a vertical overflowed edge. Traditional box expansion chooses to expand the bounding box along both *x* and *y* coordinates to resolve the overflow. However, the bounding box only needs to expand horizontally in Fig. 2.7(a) while the vertical expansion is unnecessary. Figure 2.7(b) displays the congestion map of the benchmark superblue1 after the initial routing. The red regions represent the overflowed grid edges, which normally range horizontally or vertically, implying that the situation in Fig. 2.7(a) occurs frequently during routing. Based on this observation, this work presents a novel congestion-aware bounding box expansion scheme to avoid over expanding.

Before rerouting a net, this work analyzes the amount of horizontal overflowed grid edges (HOEs) and vertical overflowed grid edges (VOEs) by tracing the routing path of the rerouted net. If the number of HOEs is more than that of VOEs, the bounding box expands vertically by δ units; on the contrary, the bounding box expands horizontally. If a tie occurs, the bounding box randomly chooses to expand horizontally or vertically. Single-direction expansion can restrict the sizes of bounding boxes to reduce the runtime. In our implement, the initial bounding box is set as the minimum rectangle enclosing two terminals to be routed, and δ is set to $5+30/r_i$, where r_i denotes the rip-up and rerouting times of the rerouted net. Moreover, based on the assumption that two opposite sides have different congestion states, extending the side near the congested region may be unnecessary, implying that the extension of each boundary of B should be discussed separately. The algorithm examines each boundary side of B to determine the necessity of box boundary expansion at the end of HUM routing. Without a loss of generality, the left boundary of B is used to illustrate the concept. Left boundary expansion can be regarded to have the intention to find a path $L_{s,t}$ on the left side of B; in addition, $L_{s,t}$ has a lower routing cost than that of the currently identified path $P_{s,t}$ (Fig. 2.7(c)). Namely, a situation in which the routing cost of $P_{s,t}$ is lower than the least cost of $L_{s,t}$ implies that the left boundary expansion is unnecessary. However, the least cost of $L_{s,t}$ is unknown because the region on the left side of B has not been explored yet. Thus, the estimated lower-bound cost of $L_{s,t}$, ec_L, is defined by the following equation to evaluate the necessity of boundary expansion. If the currently identified path $P_{s,t}$ costs less than ec_L, the left boundary remains unchanged at the next expansion of B.

$$ec_L = min_{u \in V_L, v \in V_L}(d(s,v) + d(t,u) + manh(v,u) \times \alpha) \qquad (2.4)$$

where V_L denotes the set of grid nodes on the left boundary of B; $d(s,v)$ and $d(t,u)$ represent the least cost of the unilateral monotonic routing paths from s to v and from t to u, respectively; $manh(v,u)$ refers to the Manhattan distance between v and u; and α is the lower-bound routing cost of a grid edge. In this work, α is set to 1. Notably, $d(s,v)$ and $d(t,u)$ are known values that have been computed by the HUM routing (Fig. 2.7(d)). With this, before a net n_i is rerouted, the path of n_i is first traced to obtain HOEs and VOEs. If the number of VOEs is more than that of HOEs, extending the left and right boundaries of the bounding box B of n_i is considered. If n_i is not routed by HUM routing in previous routing, the left and right boundaries of B extend immediately. Otherwise, the decision of boundary expansion is made according to the previous discussion.

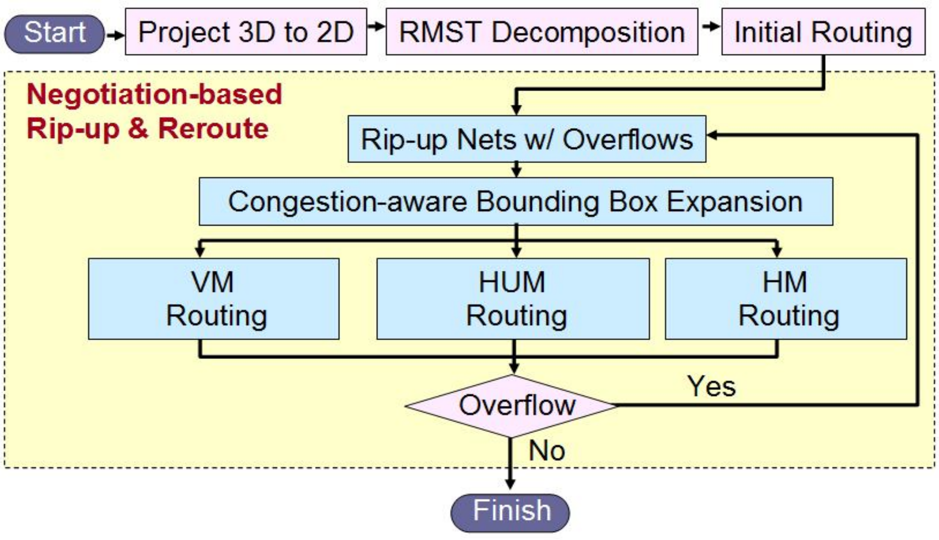


Fig. 2.8. Design flow of Grace

2.4 Design Flow of Grace

Figure 2.8 shows the design flow of the proposed routing congestion estimator Grace. First, the multi-layer routing region is projected on a 2D plan and each net is decomposed into two-pin nets based on the topology of the RMST because the works in [1, 7, 18] indicate that RMST offers better flexibility than Steiner tree to avoid blockages or congestion. An initial congestion graph is then generated by pattern routing and monotonic routing. Next, the rip-up and rerouting stage iteratively reroutes the overflowed net until an overflow-free routing result is obtained or the runtime exceeds the given time budget. In the rip-up and rerouting stage, before net n_i is rerouted, the bounding box of n_i is expanded according to the proposed congestion-aware expansion scheme. For a situation in which the width of the bounding box is equal to the x-distance between the source and the target of n_i, n_i is rerouted using HM routing. Moreover, if the height of the bounding box is equal to the y-distance between the source and the target of n_i, n_i is rerouted using VM routing. Otherwise, n_i is rerouted by HUM routing.

2.5 Experimental Results

The proposed algorithms are implemented in C/C++ language on a quad-core 2.4 GHz Intel Xeon-based linux server with a 50GB memory (only a single core is used). By hosting a routability-driven placement contest, ISPD11 has motivated many researchers to develop effective modern placers [38, 39, 42]. Ripple [38] and mPL11 placed first in the contest; their contest placement results are adopted here as the input benchmarks in our experiments. We compare Grace with NCTU-GR 2.0 [18] which is one of the fastest global routers. The experiments in [18] indicate that NCTU-GR 2.0 runs 1.90X, 1.77X and 18.66X faster than NTHU-Route 2.0 [11], FastRoute 4.1 [16], and NTUgr [12], respectively. In addition, in the old benchmarks [1, 2], the minimum wire spacing and width are uniform and all pins locate at the lowest layer. In contrast, in new benchmarks [46] used in this work, the minimum wire spacing and width of different layers are different and pins may locate at high layers. Because most of traditional routers do not consider these new features, we cannot directly adopt them to route the new benchmarks. However, recent routers NCTU-GR 2.0, BFG-R [7] and CGRIP [35] can handle these new features, but the runtime of BFG-R and CGRIP is much larger than NCTU-GR 2.0. Owing to its robustness and efficiency, the routability-driven placement contest in DAC12 [47] and ICCAD12 [48] selects NCTU-GR 2.0 to be the evaluation tool. In the following experiments, NCTU-GR 2.0 and Grace perform on the same machine. Notably, NCTU-GR 2.0 has the parameters of via cost, wirelength optimization level, pattern routing iteration, monotonic routing iteration and post routing iteration, which are set to 1, 50, 2, 2 and 0, respectively. The setting of the rip-up and rerouting stage will be detailed in following sections.

TABLE 2.1 COMPARISON TOTAL OVERFLOWS BETWEEN NCTU-GR 2.0 AND GRACE IN A GIVEN TIME BUDGET.

Benchmarks	30 seconds		60 seconds		120 seconds		240 seconds	
	NCTU	Grace	NCTU	Grace	NCTU	Grace	NCTU	Grace
s1_Ripple	14	0	0	0	0	0	0	0
s2_Ripple	724796	130866	626050	28556	472104	3798	323878	1410
s4_Ripple	66690	236	45100	222	12130	212	698	214
s5_Ripple	107896	4790	73360	404	28354	0	9030	0
s10_Ripple	597656	216364	548004	114176	491294	69646	409308	53502
s12_Ripple	315122	174162	265444	107274	180118	42604	105984	8600
s15_Ripple	92574	2664	72012	16	21048	0	5096	0
s18_Ripple	356348	175542	313146	145808	272598	130446	188594	118630
s1_mPL11	62368	0	8780	0	0	0	0	0
s2_mPL11	1152682	486966	997796	226470	857516	67202	739474	24396
s4_mPL11	112856	11010	78640	900	19334	46	4444	42
s5_mPL11	354374	71632	294312	42916	230216	27378	126398	20842
s10_mPL11	733948	271370	655464	86286	578376	29116	484624	15314
s12_mPL11	1589998	1563470	1549036	1412980	1627508	1292848	1377296	1219146
s15_mPL11	126138	0	67846	0	11410	0	0	0
s18_mPL11	66710	0	27126	0	0	0	0	0

2.5.1 Comparison between NCTU-GR 2.0 and Grace

Table 2.1 compares NCTU-GR 2.0 and Grace in terms of the ability to eliminate overflows in a given time budget. The frameworks of NCTU-GR 2.0 and Grace are nearly same; the only difference is NCTU-GR 2.0 adopts bounded-length maze routing [18] while Grace adopts unilateral monotonic routing and HUM routing in the rip-up and rerouting stage. Each major column in Table 2.1 lists the total overflows of the routing results generated by NCTU-GR 2.0 and Grace within a specified time budget for the rip-up and rerouting stage. Notably, for every benchmark, the runtime excluding the rip-up and rerouting stage is always less than 40 seconds. Table 2.1 reveals that Grace can eliminate overflows more efficiently than NCTU-GR 2.0. Given a 30 second time constraint, Grace identifies 4 overflow-free routing results while NCTU-GR 2.0 identifies none. Given a 240 second time constraint, Grace can identify 8 routing results with overflows less than a thousand, whereas NCTU-GR 2.0

identifies only 5 routing results for less than a thousand overflows. Figure 2.9 compares overflow converge curves of Grace and NCTU-GR 2.0 on benchmarks s2_mPL11, s10_mPL11, s12_Ripple and s18_Ripple, in which red curve represents NCTU-GR 2.0 while blue curve represents Grace. These figures demonstrate again that Grace can achieve fewer overflows in short runtime.

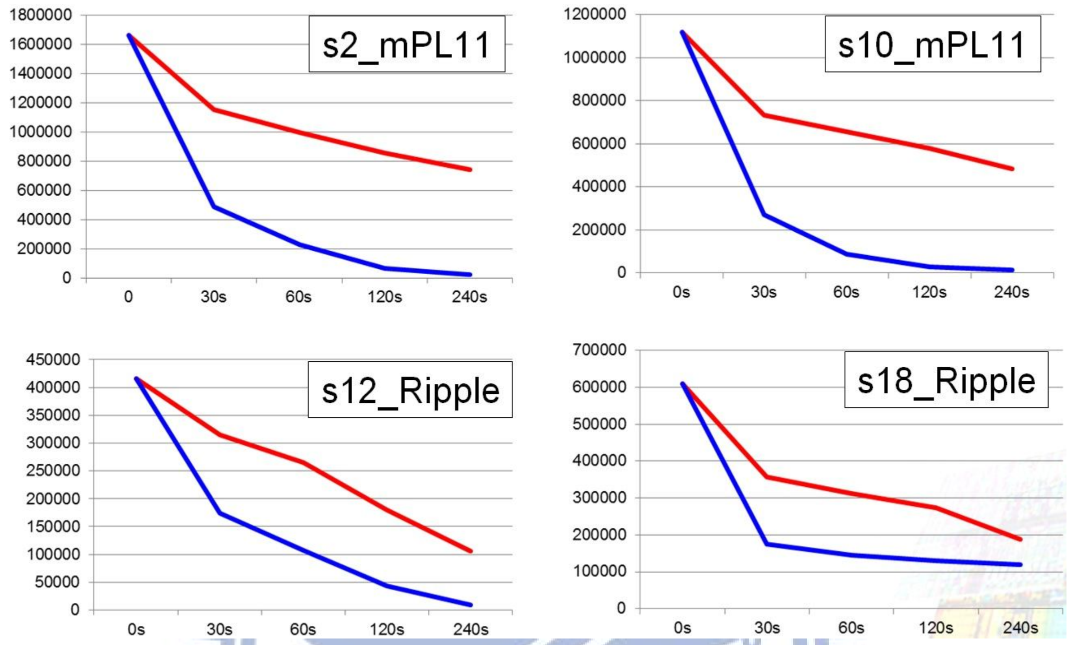


Fig. 2.9. Overflow converge curves of Grace and NCTU-GR 2.0. X-axis and y-axis denote runtime (sec) and total overflow, respectively. Red curve represents NCTU-GR 2.0 while blue curve represents Grace.

Table 2.2 compares the routing results of NCTU-GR 2.0 and Grace without a runtime limitation. Both NCTU-GR 2.0 and Grace iteratively rip-up and reroute the overflowed nets until either all overflows are eliminated or overflows cannot be reduced by more than 3% in 5 consecutive iterations. In Table 2.2, MO, TOF, WL and RCPU represent the maximum overflow, total overflows, total global routing wirelength (including wires and vias) and the runtime (seconds) of the rip-up and rerouting stage, respectively. Table 2.2 treats NCTU-GR 2.0 as a base line; $\text{Ratio}_{\text{ind}}$ denotes the average of the ratio of individual entries in the same column, while $\text{Ratio}_{\text{sum}}$ denotes the ratio of the sum of each column. To reduce the runtime, NCTU-GR 2.0 and Grace both adopt the greedy layer assignment method, explaining why the via count is not optimized. According to our experiments, the total wirelength can be further reduced by 7%~15% if the layer assignment algorithms [27] or [28] are adopted. Table 2.2 indicates that, although the wirelength and overflows of Grace are slightly worse than that of NCTU-GR 2.0, Grace can achieve a 2.75X to 26.83X higher speedup than that of NCTU-GR 2.0. Moreover, if NCTU-GR 2.0 obtains an overflow-free result for a benchmark, Grace also can achieve an overflow-free result for the same benchmark. Therefore, Table 2.2 reveals that Grace can quickly determine a placement solution whether is routable, thus making is suitable as a fast congestion estimator embedded into routability-driven placers. Figure 2.10 compares the congestion map obtained by NCTU-GR 2.0 and Grace, in which the brown regions denote the congestion regions. The congestion maps obtained by NCTU-GR 2.0 and Grace look similar.

TABLE 2.2 COMPARISON THE ROUTING RESULTS BETWEEN NCTU-GR 2.0 AND GRACE WITHOUT TIME LIMITATION.

Benchmarks	NCTU-GR 2.0				Grace				
	MOF	TOF	WL (10^5)	RCPU (s)	MOF	TOF	WL (10^5)	RCPU (s)	speedup
s1_Ripple	0	0	160.84	31	0	0	161.63	4	7.61
s2_Ripple	10	1442	350.75	3623	8	1430	358.04	203	17.81
s4_Ripple	8	224	122.80	536	6	236	124.67	30	17.75
s5_Ripple	0	0	198.44	573	0	0	202.02	86	6.67
s10_Ripple	8	39222	321.59	26509	8	40300	336.61	1227	21.61
s12_Ripple	0	0	260.86	1333	0	0	279.49	484	2.75
s15_Ripple	0	0	205.07	651	0	0	208.81	60	10.80
s18_Ripple	20	109514	159.10	5174	22	125988	162.25	550	9.40
s1_mPL11	0	0	168.81	79	0	0	169.02	9	8.66
s2_mPL11	8	3884	362.96	51356	8	4566	363.57	1914	26.83
s4_mPL11	2	42	123.89	922	2	42	126.52	94	9.76
s5_mPL11	14	11210	219.77	6114	18	15474	221.02	739	8.27
s10_mPL11	6	9678	315.31	29158	8	9370	331.49	1159	25.16
s12_mPL11	64	1041384	272.34	11221	70	1094646	259.25	1145	9.8
s15_mPL11	0	0	193.71	192	0	0	196.6	22	8.62
s18_mPL11	0	0	108.55	95	0	0	110.72	18	5.15
Ratio$_{ind}$	**1**	**1**	**1**	**1**	**1.040**	**1.089**	**1.018**	**0.114**	-
Ratio$_{sum}$	**1**	**1**	**1**	**1**	**1.071**	**1.062**	**1.019**	**0.056**	-

MOF = maximum overflow; TOF = total overflows; WL = total wirelength; RCPU = runtime of the rip-up and rerouting stage (sec)

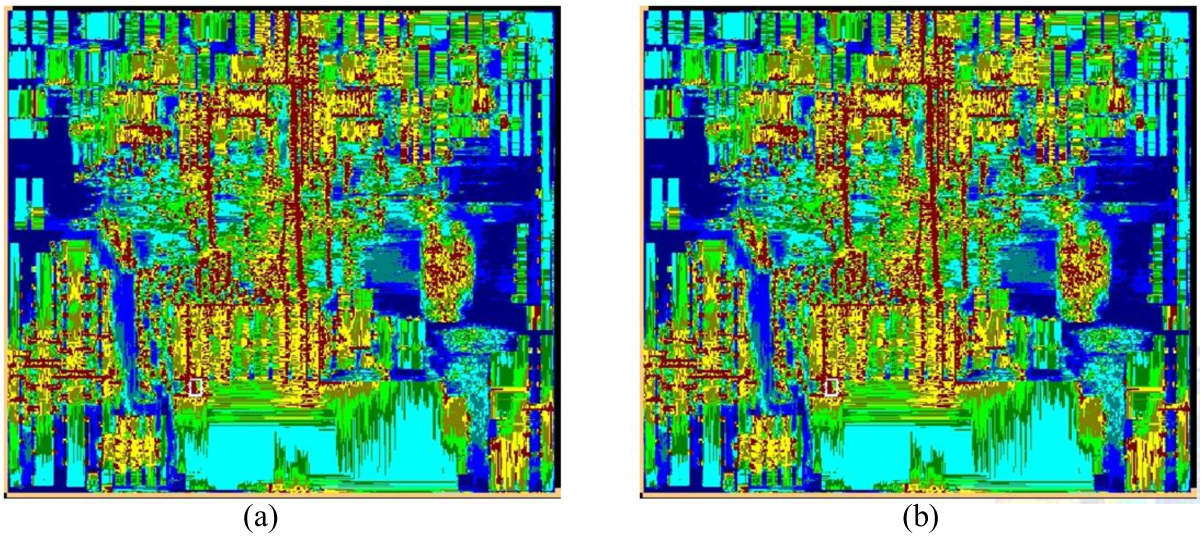

(a) (b)

Fig. 2.10. Congestion maps of s4_Ripple generated by (a) NCTU-GR 2.0 and (b) Grace.

TABLE 2.3 THE USAGE OF THE PROPOSED UNILATERAL MONOTONIC ROUTING AND HUM ROUTING ALGORITHMS

	Unila.	HUM		Unila.	HUM
s1_Ripple	15519	110	s1_mPL11	73803	3429
s2_Ripple	127654	43237	s2_mPL11	227191	286871
s4_Ripple	71128	13438	s4_mPL11	95966	86565
s5_Ripple	72995	25694	s5_mPL11	64080	178135
s10_Ripple	107671	89837	s10_mPL11	261525	104806
s12_Ripple	375051	369483	s12_mPL11	54984	1103318
s15_Ripple	79893	32905	s15_mPL11	167487	17001
s18_Ripple	62168	175323	s18_mPL11	101220	7846

2.5.2 Effectiveness of the Proposed Algorithms

Table 2.3 details the amount of usages of unilateral monotonic routing and HUM routing in Table 2.2. The column "Unila." denotes the number of two-pin nets having been routed only by unilateral monotonic routing in the rip-up and rerouting stage. The column "HUM" denotes the number of two-pin nets having been routed by HUM routing. Table 2.3 reveals that most overflows in each benchmark can be solved by unilateral monotonic routing. Because overflows in each benchmark normally range horizontally or vertically, the proposed unilateral monotonic routing can remove them efficiently. This table also indicates that most nets do not need complicated detours, so maze routing is unnecessary for these nets.

TABLE 2.4 COMPARISON BETWEEN CONGESTION-AWARE BOX EXPANSION SCHEME AND TRADTIONAL SCHEME

Benchmarks	Grace using traditional bounding box expansion scheme			
	WL	RCPU	WL imp.%	RCPU ratio
s1_Ripple	161.62	9	0.00%	1.11
s5_Ripple	201.42	234	0.29%	1.72
s12_Ripple	277.67	1890	0.65%	2.9
s15_Ripple	208.6	173	0.10%	1.87
s1_mPL11	168.84	27	0.11%	2.92
s15_mPL11	196.40	68	0.10%	2.05
s18_mPL11	110.16	46	0.51%	1.49
Average			0.25%	2.01

Table 2.4 shows the overflow-free routing results of Grace using traditional bounding box expansion scheme that is adopted by the global router in [13]. Compared to the results in Table 2.2, Grace using the traditional scheme causes roughly twice the runtime than that using the congestion-aware bounding box expansion scheme. This difference is owing to that the traditional scheme may over expand bounding boxes. However, Grace using traditional scheme can obtain the results with a slightly lower wirelength because large bounding boxes can reduce redundant detours, as demonstrated in [18].

2.6 Summary

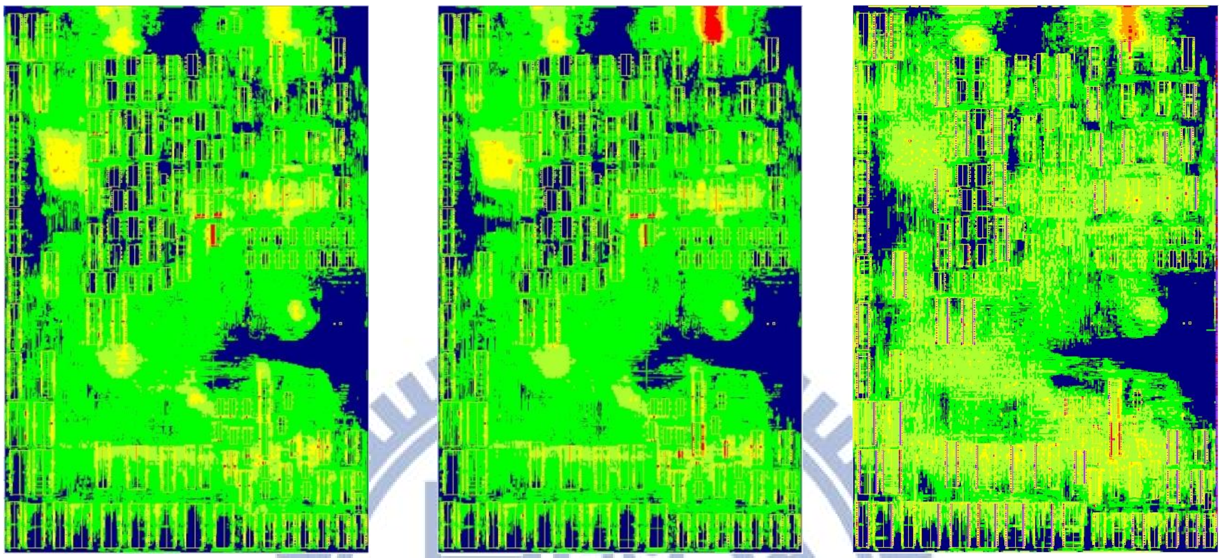
This work presents two efficient routing algorithms, unilateral monotonic routing and HUM routing. Unilateral monotonic routing can horizontally or vertically detour around the congestion regions with the same time complexity of monotonic routing. HUM routing can identify an overflow-free path on a challenged congestion map, thus making it significantly faster than maze routing with A* search scheme. Moreover, a congestion-aware bounding box expansion scheme is developed to avoid over expanding bounding boxes. Based on these contributions, a maze-free router is developed, capable of achieving 2.75X to 26.83X speedup than NCTU-GR 2.0. We believe that Grace is highly promising for use as a fast routing congestion estimator embedded into routability-driven placers.

Chapter 3 E-Grace: Enhancing Grace for Practically Industrial Designs

3.1 Introduction

In advanced technology nodes, routability has become a critical issue since considerable blockages, interconnects and complex designs rules worsen the routing congestion. To avoid wasting time on routing unroutable designs, a routing congestion estimator (RCE) can help designers to judge a design whether is routable in the early stages to speed up the design closure. Moreover, a RCE can cooperate with placers or optimizers to do optimization with consideration of routability. Therefore, the demand of a fast and accurate RCE is stringent in industry. In this dissertation, Chapter 2 presents a fast global-routing-based RCE **Grace** which is more accurate than probabilistic-based RCEs on estimating routing congestion, but Grace is still not accurate enough since timing and local congestion issues are ignored. The goal of this work is to enhance Grace to take timing and local congestion issues into account. Then enhanced Grace is named **E-Grace**.

To be aware of timing issue, the author of [49] suggests that designers formulate scenic and layer directive constraints for the timing-critical nets before feeding these nets into a router or RCE. For a net, the scenic constraint restricts its maximum routing wirelength, and the layer directive constraint specifies a range of metal layers that the net can legally route on. A net obeying the scenic constraint can avoid detouring too much and thus causing the timing violations. The layer directive constraint forces the timing-critical nets route on the higher metal layers since the wires on the higher layers have smaller delay in advanced technology nodes. Some papers [50-52] address the global routing problem with the layer directive constraint. However, how to handle the scenic constraint is seldom discussed. Figures 3.1(a) and 3.1(b) show the congestion map of E-Grace without and with consideration of the scenic constraint, and Fig. 3.1(c) shows the congestion map obtained by a congestion estimator from a full-blown industrial router which is very accurate but is much slower than E-Grace. Figure 3.1 reveals that E-Grace ignoring the scenic constraint cannot correctly estimate the congestion hot spots in the top-right corner, because the nets in Fig. 3.1(a) detour too much to dissolve the congestion. However, when the timing issue is considered, the net cannot detour so much to dissolve the congestion in the top-right corner.



(a) (b) (c)

Fig. 3.1. Congestion maps obtained by different RCEs: (a) E-Grace ignoring the scenic constraint; (b) E-Grace considering the scenic constraint; (c) a RCE from a full-blown industrial router.

In traditional global routing problem, overflow happens when a region's routing demand exceeds its routing capacity, and global routers focus on minimizing overflows. Notably, a region in global routing model usually means a G-cell. However, the duty of a RCE is more than that. For example, even given two overflow-free designs, a RCE should evaluate which design is easier to route. Accordingly, a better objective for RCE is to reduce congestion to approach a **target congestion ratio** (TCR). The **congestion ratio** of a region is the ratio of the region's routing demand to the region's routing capacity. To an industrial router, a region is easy to route if the region's maximal congestion ratio is lower than TCR, while a region's routing difficulty increases beyond linearly as the region's congestion ratio increases over TCR. Compared to minimizing overflows, we find that the routing problem becomes more complicated with the objective to minimize congestion ratio. For example, given two overflowed nets in a region with limited routing resource; we can get an overflow-free routing result when our objective is to minimize overflows. However, when the objective is to reduce congestion to approach

TCR (typically smaller than 1), the first routed net may use too many routing resource in order to meet TCR, resulting in the second routed net has no routing resource to solve its overflow, not to mention approaching TCR.

The authors of [53] indicate that local congestion causes a big mismatch between global routing and detailed routing but most global routing research [3-21] ignore the effect of local congestion. Namely, a good routing result obtained by [3-21] may not imply that a feasible detailed routing solution exists. Therefore, this work formulates the local congestion issue into E-Grace's routing model to obtain more accurate routability estimation in terms of detailed routing.

A control utility to trade off runtime and quality is important to industrial RCE tools, also users require a control utility to limit the runtime of RCEs not being crazy for over-congested designs, i.e., users hope the runtime for the same scale designs even with different congestion conditions to be similar and stable. However, the existing academic RCE tools hardly satisfy this requirement. CGRIP [37] uses a timeout constraint to be the termination condition. Since different machines have different performance, CGRIP may get nondeterministic results when it runs on different machines. BFG-R [7] and NCTU-GR 2.0 [18] set a maximum routing iteration to be the termination condition. However, the routing efforts in each iteration for designs with different congestion levels are quite different, so it is difficult to find a unique number of maximum routing iteration that works for all the designs, in order to control the runtime well.

This work presents a RCE tool E-Grace applied for industrial design flow, which includes a unified routing flow considering all the following factors: local congestion, TCR, scenic and layer directive constraints, and runtime control. This work has following contributions:

(a) A relaxation-legalization method is presented to handle the scenic constraint; it can escape from the local optimum to obtain better solution quality.

(b) This work presents a TCR-driven rip-up and rerouting (R&R) scheme, E-Grace with this scheme would solve peak congestion first in a limited runtime budget. If the runtime budget is sufficient, E-Grace would gradually reduce congestion to approach TCR.

(c) By the proposed throughput controlling method, users can trade off runtime and quality to get deterministic routing results. The runtime would linearly increase and the routing quality is monotonically improved as users increase the value of a tunable parameter

The rest of this chapter is organized as follows. Section 2 introduces the requirements and objective of a industrial RCE tool and previous works. Section 3 presents the design flow of E-Grace and the innovations in E-Grace to fulfill industrial demands. Section 4 summarizes the experimental results. Conclusions are finally drawn in Section 5.

3.2 Preliminaries

3.2.1 Problem Description

E-Grace can be treated as a very fast global router. In global routing problem, the placement is typically partitioned into an array of G-cells to form a 3- dimension (3D) grid graph $G(V, E)$, where V denotes the set of G-cells, and E refers to the set of grid edges. Each grid edge is termed by the proximity of the related G-cells to its two end nodes. The capacity $c(e)$ indicates the number of routing tracks on grid edge e. The blocked capacity $b(e)$ refers to the number of routing tracks blocked by blockages on e. The routing demand $d(e)$ indicates the number of global routing paths passing through e. The overflow of e is defined as $max(0, b(e)+d(e)-c(e))$, and most global routers focus on minimizing overflows. In this work, we redefine overflow of e as $max(0, b(e)+d(e)+w(e)-c(e))$ where $w(e)$ is the estimated number of routing tracks on e that may be used for local routes, and $w(e)$ is obtained by the method in [53].

Rather than minimizing overflows, the objective of this work is to efficiently minimize congestion to approach TCR as close as possible. This work sets TCR to 80% because, in the industrial environment where we test E-Grace, 20% of the routing capacity is reserved to route non-signal nets such as clock nets later in the design flow. Moreover, each net in the routing result has to obey the scenic and layer directive constraints. The scenic constraint restricts that the wirelength of each net n cannot exceed an upper bound $B(n)$, and the layer directive constraint restricts that n only can route on the layers between *tl*(*n*) and *bl*(*n*) where *tl*(*n*) and *bl*(*n*) can be any layer among all layers, but *tl*(*n*)>*bl*(*n*). In addition, to satisfy the industrial demands, a control utility to trade off the runtime and routing quality of E-Grace is required.

3.2.2 Congestion Evaluation Metrics

This work adopts a net-based metric and a grid-edge-based metric to evaluate the routing congestion. Given a routing result, we first calculate the congestion ratio *g*(*e*) for each grid edge *e* by the following equation,

$$g(e) = [d(e) + b(e) + w(e)]/c(e)\,. \qquad (3.1)$$

After that, net-based congestion metric *WCI*(*x*) is defined as the number of nets whose congestion is greater or equal to *x*%, the congestion of a net is the maximum congestion ratio among all grid edges traversed by the net. Moreover, grid-edge-based congestion metric *ACE*(*y*) is computed by averaging the congestion ratio of the top *y*% congested grid edges. In our experiments, we set $y \in \{0.5, 1, 2, 5, 10, 20\}$. *ACE*(0.5) provides a local view for the peak congestion, while *ACE*(20) provides a global view for the average congestion.

3.2.3 Previous Works

Grace is a fast global-routing-based RCE tool, which relies on two efficient routing algorithms called unilateral monotonic routing and HUM routing, and invokes a congestion-aware bounding box expansion scheme to avoid over-expansion. The routing kernel of E-Grace is similar to Grace. However, Grace cannot handle scenic and layer directive constraints, and does not consider the issues of local congestion and TCR.

Grace adopts a 2D routing with layer assignment framework to solve 3D global routing problem. However, this framework may struggle for the layer directive constraint because it has no precise layer information during 2D routing. For example, a design has 9 layers and a region has congestion between layers 5 and 9. A net with layer directive constraint {1, 9} can legally pass through this region, but a net with layer directive constraint {7, 8} would suffer congestion when it passes through this region. However, the 2D routing stage in Grace cannot distinguish these two cases.

The layer-directive-aware global router GLADE [50, 51] maintains a virtual demand data structure during 2D routing to query the estimated 3D congestion between two specified layers. In contrast, the work in [52] adopts a grouping method to handle the layer directive constraint, which classifies the nets with the same layer directive constraint into a group, and then sorts each group in increasing order according to the range of each group's layer directive. Next, each group is sequentially processed during 2D routing. For each group, only one layer directive constraint needs to be addressed, which simplifies the problem. Although GLADE [50, 51] can explore larger solution space to get better results, [52] is faster. Due to the runtime consideration, the grouping method [52] is adopted in this work.

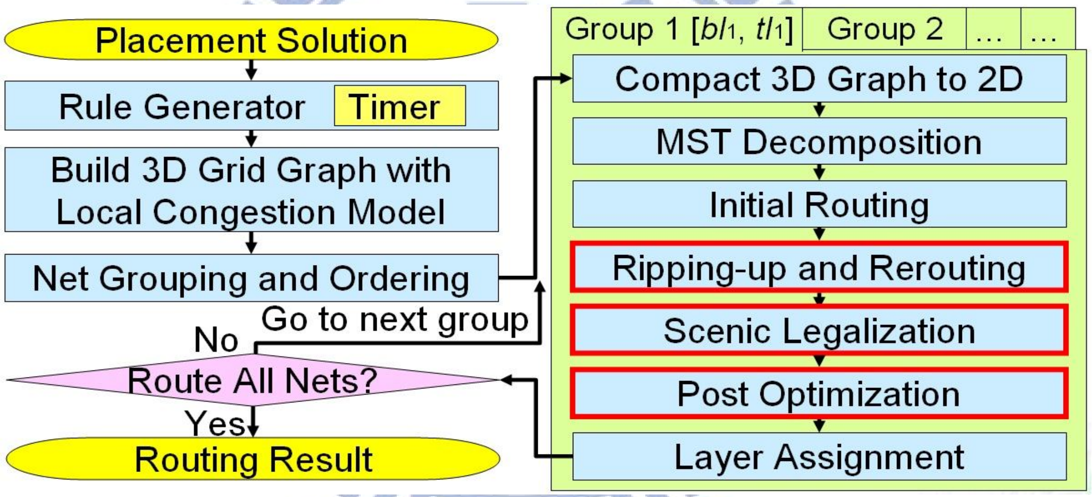


Fig. 3.2. Design flow of E-Grace

3.3 Design Flow of E-Grace

Figure 3.2 shows the design flow of E-Grace, in which the red boxes highlight the stages including the innovations in this work. At first, given a placement solution and a netlist, a rule generator is invoked to generate the layer directive and scenic constraints for each net. The rule generator uses an industrial timer to analyze the timing slack for each net, and then the method in [54] is used to assign layer directive constraints for the timing-critical nets. Moreover, for each net n, the scenic constraint $B(n)$ is identified by each net's RMST length multiplying a variable p_n that is between 1.05 and 1.4.

Typically, p_n of most nets (>90%) is around 1.15 in the industrial environment used in this work and some timing-critical nets have smaller p_n. Because the rule generator is not the focus in this work, the details of the rule generator are skipped. After that, a 3D grid graph is built and the local-congestion-aware factor $w(e)$ in Eq. (3.1) is calculated by the pin-density method in [53].

Before performing routing, the grouping method [52] is used to classify nets into several groups and then sorts groups. Subsequently, E-Grace routes nets group by group. At the beginning of processing a group with layer directive $\{bl_i, tl_i\}$, the partial 3D grid graph from layer bl_i to tl_i is projected on the 2D grid graph and each net in this group is decomposed into two-pin subnets based on the topology of the RMST. Note that, we call a two-pin subnet as a ***segment*** in this work. If a segment s belongs to a net n, n is the ***parent net*** of s. In this work, we use RMST instead of RSMT to decompose the nets, because the works in [1, 7, 18] indicate that RMST offers better flexibility than RSMT to avoid blockages or congestion. Also, we found that the runtime and wirelength of FLUTE [23] (a well-known RSMT generator) are worse than RMST for some modern industrial designs, because these designs contain considerable high fan-out nets and RMST runs faster and produce shorter wirelength for very high fan-out nets than FLUTE. Accordingly, E-Grace adopts the algorithm in [55] to efficiently build RMST. After that, the initial routing stage generates an initial routing result by pattern routing and monotonic routing. Because pattern routing and monotonic routing do not make detours, every net in the initial routing result must obey the scenic constraint. Next, the ripping-up and rerouting (R&R) stage iteratively reroutes the congested segments until either the congestion of any net is not greater than TCR or the termination condition is satisfied. The definition of congested segments and the termination condition will be introduced later.

In the R&R stage, the scenic constraint is relaxed as a soft constraint, namely the routing solutions in this stage can violate the scenic constraint. Then, the scenic legalization stage reroutes the nets with the scenic violation to force them to satisfy the scenic constraint, but this stage may increase congestion. In the post optimization stage, the congested segments are rerouted under the hard scenic constraint to reduce the congestion worsened by the scenic legalization stage. Using this relaxation-legalization method to handle the scenic constraint can get better results than always treating the scenic constraint as a hard constraint. Finally, we adopt a fast heuristic layer assignment algorithm presented in [17] to map the 2D routing result to the 3D grid graph. E-Grace sequentially processes each group until the routing solutions of all nets are obtained.

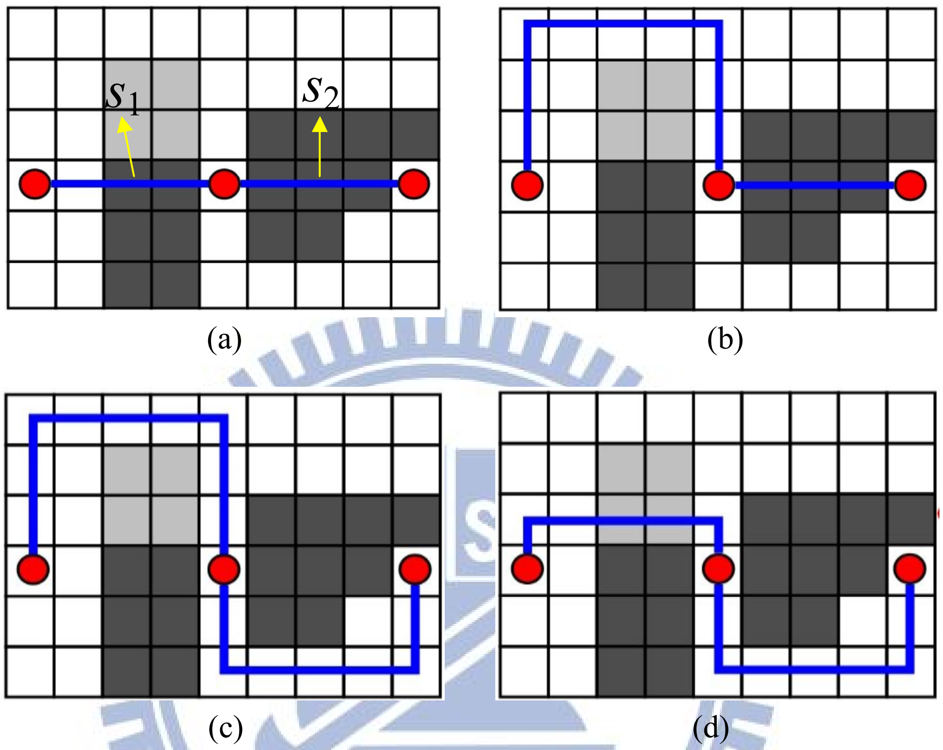


Fig. 3.3. (a) the initial routing result of a net; (b) a segment exhausts all detour quotas, resulting another segment has no detour quotas to bypass congestion regions; (c) the routing result of the proposed R&R stage; (d) the routing result after the scenic legalization stage.

3.3.1 Relaxation-Legalization Scenic Controlling Method

Most global routers decompose multi-pin nets into several segments and then route each segment individually, since it can reduce the multi-terminal routing problem to a two-point routing problem. However, the scenic constraint is not easy to meet when using this method. For instance, Fig. 3.3(a) shows an initial routing result of a three-pin net n with two segments s_1 and s_2, in which the light and dark gray rectangles respectively denote the light congestion and very congestion regions. The wirelength of n is 8 and the scenic constraint restricts that the wirelength of n cannot exceed 14, namely n has 6 detour quotas. If s_1 is routed earlier than s_2 and is allowed to use all detour quotas, we will get a routing result shown in Fig. 3.3(b), in which s_2 cannot bypass the congestion regions because the detour quotas have run out. Suppose we averagely allocate the detour quotas to s_1 and s_2 to avoid s_1 exhausting all detour quotas, both s_1 and s_2 cannot bypass the congestion regions because they both have no enough detour quotas.

This work presents a relaxation-legalization method to handle the scenic constraint. At first, we allow the routing solutions with scenic violations for reducing congestion. Then, we dynamically adjust the routing cost function to encourage the routing solutions gradually fitting the scenic constraint. In this work, $r_c(e,s)$ denotes the routing cost of grid edge e for segment s, which consists of congestion cost $c_c(e)$ and wirelength cost $w_c(s)$. Assuming the initial routing solution of s passes through e and a least-cost routing algorithm is adopted to reroute s. If $c_c(e)$ dominates $r_c(e,s)$, the algorithm may identify a detoured path to bypass the congestion on e. If $w_c(s)$ dominates $r_c(e,s)$, the algorithm may identify a path passing through e to save wirelength. Accordingly, we can adjust the ratio between $c_c(e)$ and $w_c(s)$ to indirectly control the routing wirelength of s. Notably, the least-cost routing algorithm here can be any shortest path finding algorithm such like A* search or Dijkstra's algorithm, while this work uses unilateral monotonic and HUM routing.

For the case in Fig. 3.3(a), E-Grace first generates a routing solution with the scenic violation for reducing congestion (Fig. 3.3(c)). Next, E-Grace iteratively reroutes s_1 and s_2, and gradually increases $w_c(s_1)$ and $w_c(s_2)$ to drive them to route shorter until net n satisfies its scenic constraint. In our example, because s_1 faces a lighter congestion region than s_2, driving s_1 to take the shorter path is easier than driving s_2. Thus, we can get the routing result in Fig. 3.3(d).

3.3.1.1 Ripping-up and Rerouting Stage

This stage treats the scenic constraint as a soft constraint; each net can violate its scenic constraint to solve congestion. However, too many nets with scenic violations would make the subsequent scenic legalization stage sacrifice congestion a lot to satisfy the scenic constraint. Accordingly, if the parent net n of segment s has scenic violations, this stage increases $w_c(s)$ before rerouting s to prevent the new routing solution violating the scenic constraint again. This stage formulates $w_c(s)$ as follows:

$$w_c(s) = \alpha + \beta \times (v_s)^2 + \gamma \times (v_n)^2 , \qquad (3.2)$$

where α, β and γ are user defined constants, we set them to be 500, 20 and 50 in our implementation, respectively. Notations v_n and v_s initially are zero. Once the wirelength of n exceeds $B(n)$ after rerouting s, v_n increases by one. Once the wirelength of s exceeds $b(s)$ after rerouting s, v_s increases by one, where $b(s)$ is the suggested wirelength for s, that is linearly proportional to the ratio of initial wirelength of s and n, where the initial wirelength means the wirelength of a net or a segment at the beginning of the R&R stage. For example, $B(n)$ is 30 and n consists of three segments s_1, s_2 and s_3 whose initial wirelength are 6, 10 and 4, respectively. The $b(s_1)$, $b(s_2)$ and $b(s_3)$ are 9 (=6*(30)/(6+10+4)), 15 (=10*3/2) and 6 (=4*3/2), respectively.

In this stage, $c_c(e)$ also increases if e frequently becomes congested, which makes the least-cost routing algorithms more actively bypass e to reduce congestion. So, we cannot guarantee that the routing wirelength of s always becomes shorter when $w_c(s)$ and $c_c(e)$ both are increased. The formulation of $c_c(e)$ will be introduced in section 3.2.

3.3.1.2 Scenic Legalization Stage

This stage consists of the soft and hard legalization phases. At the beginning of the soft legalization, an array is built to contain all segments whose parent net violates the scenic constraint. Then, the array is iteratively scanned to check each segment s in the array. If the parent net of s has the scenic violation and the wirelength of s is longer than the Manhattan distance between its two terminals, we increase $w_c(s)$ and then reroute s. In this stage, $c_c(e)$ would not continually increase even e frequently becomes congested, so we can guarantee that the routing wirelength of s monotonically decreases as s is rerouted and $w_c(s)$ is increased.

The goal of the soft legalization is to reduce the number of nets with scenic violations in a few iterations but does not sacrifice congestion too much. How to properly increase $w_c(s)$ is challenging. Increasing $w_c(s)$ too much would worsen congestion a lot. If the increment of $w_c(s)$ compared to $c_c(e)$ is slight, the reduction of the scenic violations is also slight in each iteration.

The soft legalization uses a cost accumulation method to update $w_c(s)$ properly. Before rerouting s, we trace the path of s and compute the path cost of s, *pathCost*(s), which is the sum of the routing cost of each grid edge passed by the path of s, and then $w_c(s)$ is updated by the following equation.

$$w_c(s) = \lceil pathCost(s)/manh(s) \rceil, \qquad (3.3)$$

where *manh*(s) denotes the Manhattan distance between the two terminals of s. Fig. 3.4 shows an example to illustrate how Eq. (3.3) works. Figure 3.4(a) shows a routing solution of s with the scenic violation at the beginning of the soft legalization, in which $w_c(s)$ is 1. Assume $c_c(e)$ is 4 for edge e which is light congested (light grey edges in the figure); if e is very congested (dark grey edges), $c_c(e)$ is 10; otherwise (white edges), $c_c(e)$ is 0. Before rerouting s, we trace the path of s in Fig. 3.4(a) to obtain *pathCost*(s) that is 10. Based on Eq. (3.3), $w_c(s)$ is updated to 2.5 (=10/4) and then rerouting s with new $w_c(s)$ obtains the routing solution in Fig. 3.4(b). If the routing solution in Fig. 3.4(b) still violates the scenic constraint, $w_c(s)$ is updated to 6 (=(8*2.5+4)/4) and then s is rerouted again in the next iteration. After that, the routing solution in Fig. 3.4(c) is obtained. The example in Fig. 3.4 reveals that the wirelength of s monotonically decreases as the routing iterations increases.

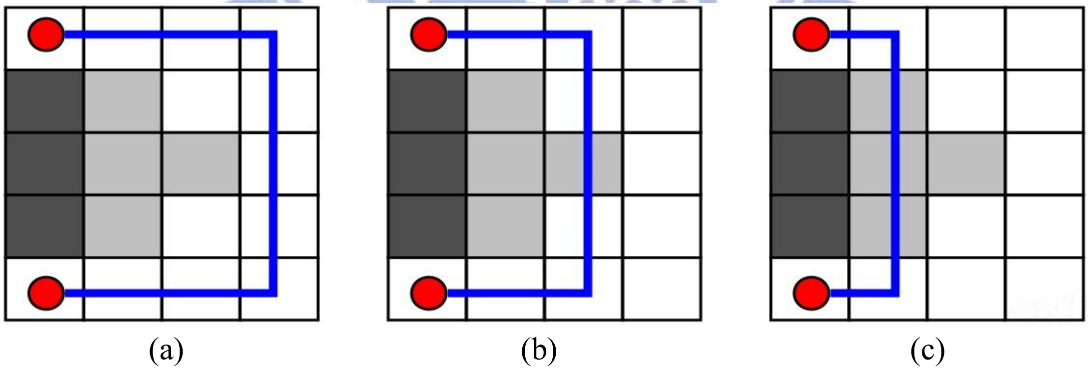


Fig. 3.4. The routing result obtained by (a) the R&R stage; (b) the first iteration of the soft legalization phase; (c) the second iteration of the soft legalization phase.

In our implementation, the soft legalization performs three iterations, which can eliminate most scenic violations. However, there are few remained scenic violations that need to be solved by the hard legalization. In the hard legalization phase, we adopt the monotonic routing to reroute the segments whose parent net has the scenic violation. Because monotonic routing makes no detour, a net must conform to its scenic constraint when all segments of the net are routed by monotonic routing. After the hard legalization phase, all nets must obey the scenic constraints.

3.3.1.3 Post Refinement Stage

The goal of the post refinement stage is to reroute the congested segments to reduce the congestion worsened by the scenic legalization stage. For example, after the R&R stage, two neighboring segments s_1 and s_2 both have congestion-free routing paths but the parent net of s_1 violates the scenic constraint. After the scenic legalization stage, s_1 is rerouted to be shorter, in which causes s_1 now to compete the routing resource with s_2 and thus causes the congestion. In this case, rerouting s_2 may dissolve the congestion.

In the post refinement stage, the routing path of each segment is traced once. A segment is rerouted if the segment passes through a congestion region and has not been rerouted in the scenic legalization stage. To ensure that the routing solutions in this stage always obey the scenic constraint, if the new routing solution obtained by rerouting violates the scenic constraint, we discard the new routing solution and restore the segment to the old routing solution.

3.3.2 TCR-driven R&R Scheme

The problem of minimizing congestion to approach TCR (80% in this work) is more complicated than minimizing overflows. For example, given a limited runtime budget, a RCE may spend most time on rerouting the nets whose congestion is between 80% and 100% to meet TCR, finally has no time to reroute the nets whose congestion is over 100%. In another case, given a region with limited routing resources, the early routed nets may consume too many routing resources when they target to meet TCR, causing the later routed nets have no routing resources to reduce congestion. Based on above cases, we can summarize two requirements for TCR-driven RCE: (1) route the nets with peak congestion first in the limited runtime budget; (2) reduce congestion averagely. For example, two nets with 90% congestion are better than one with 80% and another with 100%.

The R&R stage adopts a TCR-driven R&R scheme to satisfy abovementioned two requirements. In the R&R stage, if a segment passes through a grid edge e satisfying the following inequation, the segment is treated as a **congested segment**. The congested segment will be rip-upped and rerouted.

$$g(e) > \max(1 - p \times \lambda, \mathrm{TCR}) \,, \tag{3.4}$$

where p is an iteration count and initially is zero, p increases by one as iteration increases; λ is a user defined constant, we set λ to 0.02. According to Eq. (3.4), the first iteration of the R&R stage would reroute the nets whose congestion is greater than 100%, the second iteration would reroute the nets whose congestion is greater than 98%, and so on. The R&R stage iteratively reroutes the congested segments until either no net has congestion greater than TCR or termination condition meets. This R&R scheme can solve the peak congestion first to meet requirement (1).

To meet requirement (2), we formulate the congestion cost $c_c(e)$ as follows:

$$\begin{aligned} c_c(e) &= [1+\frac{C_1}{1+{C_2}^{C_3 \times to(e)}} + C_4 \times (o(e))^2] \times [1+(\mathrm{h}_e)^2] \\ o(e) &= \max(0, d(e)+w(e)+b(e)-c(e)) \\ to(e) &= d(e)+w(e)-\max(0, c(e)\times \mathrm{TCR} - b(e)) \end{aligned} \,, \tag{3.5}$$

where C_1, C_2, C_3 and C_4 are user defined constants, we set them to 200, 2.72, -0.3 and 5, respectively. History cost h_e is initialized to zero for each grid edge e at the beginning of the R&R stage. If the routing solution of s passes through e after rerouting s and $to(e)$>0, h_e increases by one. Notably, $to(e)$>0 means that the congestion ratio of e is greater than TCR. Figure 3.5 shows that the curve of $c_c(e)$ when $c(e)$, $w(e)$, $b(e)$, TCR and h_e are 50, 0, 0, 80% and 0, in which x-axis and y-axis respectively denotes $d(e)$ and $c_c(e)$. Using Eq. (3.5) to formulate $c_c(e)$ encourages that routing algorithms avoid overflows but do not use too much routing resource for pushing congestion down to TCR. However, if the congestion of e is continuously over TCR in the R&R stage, $c_c(e)$ gradually increases since h_e increases. This makes the

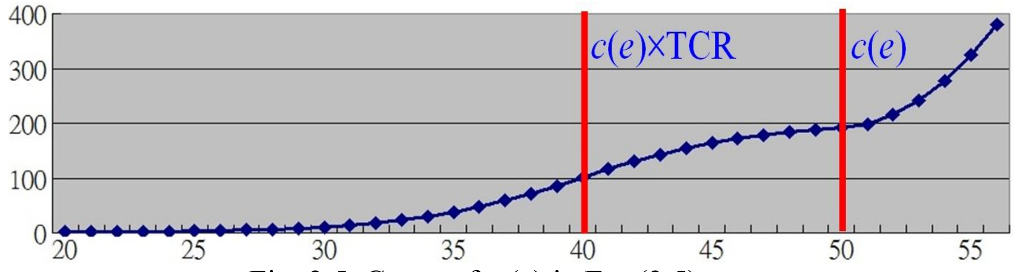


Fig. 3.5. Curve of $c_c(e)$ in Eq. (3.5).

routing algorithms become more active to push congestion down to TCR when the routing iteration increases. Notably, h_e would not increase in the scenic legalization and post refinement stages.

3.3.3 Throughput Controlling

A control utility to trade off runtime and quality is essential to an industrial RCE tool. Based on the feedback from users, we summarize three features that an industrial RCE tool requires: (1) when users increase the value of a tunable parameter t_u, the congestion in the routing result should monotonically improve and the runtime of the RCE tool linearly increases; (2) with the same value of t_u, the RCE tool can automatically give a design more runtime budget to route if the design is larger (with more nets or larger routing grid size); (3) with the same value of t_u, the runtime of a RCE tool can be stable for any congestion circumstance. Because, a RCE tool may cooperate with other tools to optimize a design, but other tools may worsen congestion when they address other issues. Feature (3) ensures that the runtime of the optimization flow can be stable and predictable.

The R&R stage mainly influences the runtime and the routing quality in E-Grace; hence this work presents a throughput controlling method to set a termination condition for the R&R stage. In E-Grace, the R&R stage for group R terminates if the following condition holds.

$$tvg > t_u \times \mu \times \sum_{s \in S_R} numGcell(s), \qquad (3.6)$$

where t_u is the tunable parameter to control the tradeoff between runtime and quality, S_R denotes the set of all segments in group R, and μ is a user defined constant. In our implementation, u is set to 0.5. Moreover, $numGcell(s)$ denotes the number of G-cells within the initial bounding box of s which is the minimum rectangle enclosing the terminals of s. The right term in Eq. (3.6) can be initialized in the RMST decomposition stage. Notation tvg denotes the number of visited G-cells in the R&R stage. Initially, the value of tvg is zero. During the R&R stage, when the path of a segment is traced, tvg increases by the length of the traced path. If the segment is a congested segment, the bounding box of the segment will be expanded and then the segment will be rerouted by the unilateral monotonic and HUM routing algorithms within the bounding box. Because the time complexity of unilateral monotonic and HUM routing is linear to the number of G-cells in the bounding box, *tvg* increases by the number of G-cells in the bounding box after a segment is rerouted. We can treat *tvg* as an indicator for the computation throughput, and we can adjust t_u to control the upper bound of the computation throughput and easily satisfy the abovementioned three features.

TABLE 3.1 DESIGN INFORMATION

	Grid size	#Nets(10^5)		Grid size	#Nets(10^5)		Grid size	#Nets(10^5)
Ind1	357×535×9	13.3	**Ind3**	1068×710×10	41.7	**Ind5**	444×518×10	16.6
Ind2	1068×710×10	40.5	**Ind4**	227×740×11	16.7	**Ind6**	534×407×10	9.2

3.4 Experimental Results

The proposed algorithms are implemented in C/C++ on a linux server with four 2.27 GHz Xeon E7-8860 CPUs. This work compares E-Grace (EG_{all}) with a fast industrial congestion estimator (CA_{ind}) and a full-blown congestion estimator (GR_{ind}) from an industrial router. Table 3.1 shows the grid size and net number of the industrial designs used in the following experiments, in which Ind2 and Ind3 are from the same design but their placements are totally different.

3.4.1 Compare E-Grace with other industrial RCE tools

In Table 3.2, EG_{all} includes all innovations presented in this work; we set parameter t_u in Eq. (3.6) to 5 to strike a good balance between the routing quality and runtime. The effectiveness of individual innovations in this work will be detailed later. CA_{ind}, similar to that in [53], is a fast global-routing-based RCE based on the edge-shifting algorithm [8] to avoid invoking time-consuming maze routing algorithms. The edge-shifting method is fast but may struggle for the very congested situations because the solution space of the edge-shifting method is restricted. GR_{ind}, with a complex local congestion model to consider the local routability, runs most accurate and will be used to judge the quality of all results but is much slower than EG_{all} and CA_{ind}. Notably, CA_{ind} and GR_{ind} both can handle layer directive and scenic constraints, and take the local congestion and TCR issues into account. In terms of the scenic constraint, CA_{ind} always treats the scenic constraint as a hard constraint, and the situation where a segment may exhaust all detour quotas like the example in Fig. 3.3(b) may happen. In contrast, GR_{ind} gradually relaxes the scenic constraint for the rerouted nets, so the routing results of GR_{ind} may violate the scenic constraint.

TABLE 3.2 ROUTING RESULTS COMPARISON BETWEEN E-GRACE, INDUSTRIAL CONGESTION ANALYZER AND REAL ROUTER

		Wall time(s)	WCI(100)	WCI(90)	ACE(0.5)	ACE(1)	ACE(5)	ACE(10)	ACE(20)	WL (10^7)
Ind1	EG_{all}	188	2021	18781	92.93	88.88	82.18	80.84	79.47	27.03
	CA_{ind}	368	13736	46925	98.30	93.49	85.05	82.41	80.17	26.94
	GR_{ind}	21222	1047	14335	88.66	86.27	82.85	81.64	80.32	27.05
Ind2	EG_{all}	436	767	52226	92.70	90.90	83.72	80.83	76.92	16.01
	CA_{ind}	1987	6845	42893	94.24	91.54	83.99	80.86	78.47	16.30
	GR_{ind}	14649	41	10514	87.66	86.34	82.98	80.83	76.08	16.08
Ind3	EG_{all}	422	4257	166705	95.98	94.34	88.62	84.69	80.90	18.27
	CA_{ind}	1697	16055	157971	97.34	95.37	89.09	85.38	81.40	17.90
	GR_{ind}	10376	895	29303	89.96	87.92	84.20	82.48	79.21	17.93
Ind4	EG_{all}	101	27	44100	92.68	90.95	84.27	81.46	74.48	8.14
	CA_{ind}	162	22747	78090	101.57	98.62	88.20	83.70	76.12	7.99
	GR_{ind}	11064	104	9543	87.93	86.05	81.24	78.01	69.22	8.42
Ind5	EG_{all}	140	149	130556	93.80	92.58	89.07	86.85	83.83	11.82
	CA_{ind}	216	4123	277376	95.03	93.77	90.59	87.91	84.32	12.05
	GR_{ind}	47261	1141	89904	94.62	93.48	90.91	88.82	85.66	11.39
Ind6	EG_{all}	51	1	1598	88.47	85.32	80.63	78.08	71.46	4.69
	CA_{ind}	108	1478	3309	92.38	87.44	80.24	78.11	72.11	4.54
	GR_{ind}	1672	19	2788	86.33	84.77	80.63	76.61	67.44	4.56
Ratio	EG_{all}	1	1	1	1	1	1	1	1	1
	CA_{ind}	2.633	394.608	1.706	1.040	1.032	1.017	1.011	1.012	0.994
	GR_{ind}	108.476	5.215	0.632	0.962	0.967	0.989	0.991	0.979	0.993

Table 3.2 adopts a net-based metric WCI and a grid-edge-based metric ACE to evaluate the routing results of EG_{all}, CA_{ind} and GR_{ind}, in which the unit of ACE is %. Table 3.2 shows that EG_{all} respectively runs 2.6× and 108.5× faster than CA_{ind} and GR_{ind} on average. (EG_{all} and CA_{ind} use one thread, while GR_{ind} uses four threads). In addition, the routing results obtained by EG_{all} have better congestion than that obtained by CA_{ind}, and the congestion analysis of the routing results obtained by EG_{all} is similar to

that of GR_{ind}. Table 3.2 reveals that EG_{all} can achieve fast and accurate congestion estimation. Fig. 3.6 shows the congestion maps obtained by EG_{all}, CA_{ind} and GR_{ind} for design Ind4.

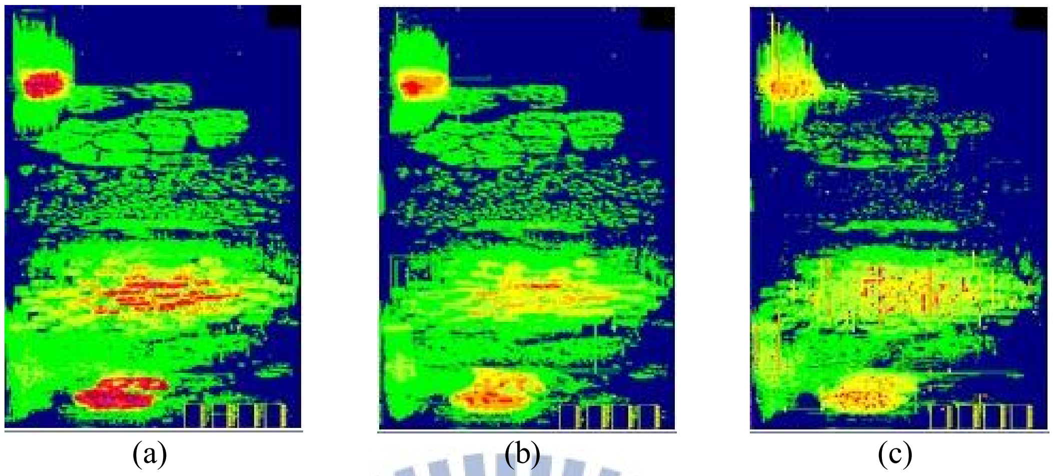


Fig. 3.6. Congestions maps of Ind4 obtained by (a) CA_{ind}, (b) EG_{all}, (c) GR_{ind}.

3.4.2 Effectiveness of each Innovation in E-Grace

To evaluate individual innovations in this work, we run more detailed experiments in this section. However, due to the licensing issue, the following experiments cannot adopt the designs shown in Table 3.1. Therefore, the following experiments adopt another suit of industrial designs shown in Table 3.3 and perform on a 2.4 GHz Xeon-based linux server with E5620 CPU.

TABLE 3.3 DESIGN INFORMATION

	Grid size	#Nets(10^5)		Grid size	#Nets(10^5)		Grid size	#Nets(10^5)
Ind7	800×415×10	9.0	**Ind9**	704×413×10	10.6	**Ind11**	444×518×10	16.6
Ind8	774×570×10	8.0	**Ind10**	357×535×9	13.3	**Ind12**	425×516×10	13.9

3.4.2.1 Effectiveness of Relaxation-Legalization

To evaluate the effectiveness of each stage in relaxation-legalization method, we built several versions of E-Grace to handle the scenic constraint. Table 3.4 lists the differences between each version, in which EG_1 does not have scenic legalization and post refinement stages, and set β and γ in Eq. (3.2) to zero to totally ignore the scenic constraint in its entire flow. In contrast, EG_2, EG_3 and EG_{all} use the different levels of the relaxation-legalization method to handle the scenic constraint. Notably, t_u in Eq. (3.6) is set to 5 for each version.

TABLE 3.4 DIFFERENT VERSIONS OF E-GRACE

EG$_1$	without consideration of the scenic constraint
EG$_2$	EG$_1$ + Scenic legalization
EG$_3$	EG$_2$ + Default Eq. (3.2)
EG$_{all}$	EG$_3$ + Post Refinement

TABLE 3.5 EFFECTIVENESS OF USING RELAXATION-LEGALIZATION METHOD TO HANDLE THE SCENIC CONSTRAINT

		CPU(s)	WCI(100)	WCI(90)	ACE(0.5)	ACE(1)	ACE(2)	ACE(5)	ACE(10)	ACE(20)	WL(10^7)	SV
Ind7	EG$_1$	40.42	1182	26950	95.50	92.92	90.36	85.51	82.60	79.71	11.18	4102
	EG$_2$	46.29	4307	25206	97.63	94.45	91.03	85.46	82.51	79.40	11.06	0
	EG$_3$	42.46	3554	25367	97.19	94.09	90.82	85.40	82.49	79.41	11.06	0
	EG$_{all}$	45.63	3257	24216	96.61	93.52	90.25	85.03	82.30	79.29	11.06	0
Ind8	EG$_1$	43.19	2660	22103	98.25	94.58	91.14	85.27	82.02	76.85	10.94	4368
	EG$_2$	44.12	5889	21407	99.08	95.88	91.63	85.13	81.72	76.34	10.83	0
	EG$_3$	43.86	5556	21580	98.77	95.65	91.55	85.14	81.74	76.38	10.84	0
	EG$_{all}$	44.54	5000	18985	97.93	94.75	90.69	84.61	81.42	76.14	10.83	0
Ind9	EG$_1$	30.61	752	9401	103.36	97.01	91.23	84.76	81.74	77.31	10.73	1909
	EG$_2$	36.16	2305	10494	98.85	94.36	89.37	83.89	81.21	76.79	10.66	0
	EG$_3$	31.13	2106	10488	98.89	94.34	89.35	83.89	81.20	76.78	10.66	0
	EG$_{all}$	32.29	1098	9383	97.31	93.76	89.24	83.87	81.22	76.87	10.67	0
Ind10	EG$_1$	84.31	1343	13384	92.22	89.31	86.70	83.31	81.47	79.58	24.15	3883
	EG$_2$	86.50	1788	16781	92.75	89.63	86.80	83.30	81.45	79.52	24.09	0
	EG$_3$	85.89	1491	16683	92.61	89.53	86.75	83.29	81.44	79.52	24.10	0
	EG$_{all}$	85.04	1681	13589	91.88	88.60	85.80	82.66	81.10	79.34	24.13	0
Ind11	EG$_1$	43.24	157	86159	94.23	92.65	91.49	89.63	87.15	83.81	15.14	10955
	EG$_2$	45.33	123	77926	94.6	92.84	91.58	89.42	86.67	83.35	14.78	0
	EG$_3$	44.08	123	77921	94.59	92.84	91.58	89.42	86.67	83.35	14.78	0
	EG$_{all}$	45.55	77	77080	93.82	92.41	91.32	89.23	86.56	83.29	14.78	0
Ind12	EG$_1$	27.07	4359	52158	103.23	99.35	95.89	91.18	87.48	83.58	12.75	5286
	EG$_2$	28.81	5353	51490	99.82	97.93	95.40	91.03	87.29	83.42	12.58	0
	EG$_3$	28.36	5162	51690	99.83	97.90	95.33	90.99	87.26	83.41	12.58	0
	EG$_{all}$	30.49	4480	50166	99.74	97.84	95.26	90.83	86.99	83.24	12.59	0
Ratio	EG$_1$	1	1	1	1	1	1	1	1	1	1	
	EG$_2$	1.081	2.044	1.028	0.994	0.999	0.998	0.997	0.997	0.996	0.989	
	EG$_3$	1.028	1.829	1.029	0.992	0.998	0.997	0.997	0.997	0.996	0.989	
	EG$_{all}$	1.067	1.478	0.938	0.985	0.992	0.992	0.993	0.994	0.994	0.989	

Table 3.5 shows the routing results obtained by each version, in which SV denotes the number of nets with the scenic violation. The scenic legalization stage in EG_2 can eliminate all scenic violations but worsens the congestion. Averagely, EG_2 gets better ACE values but worse WCI(100) than EG_1. Although EG_2 for design Ind11 obtains the results with better WCI(100) than EG_1, EG_2 has worse ACE(0.5) than that of EG_1. This phenomenon implies that only relying on either net-based metric or grid-edge-based metric to evaluate the congestion may get a biased view, while using both metrics provides a more comprehensive view.

EG_3 has the consideration of the scenic constraint in the R&R stage, which reduces the number of nets that need to be legalized in the scenic legalization stage. Hence, the runtime and overhead of congestion degradation in the scenic legalization diminish. Compared to EG_2, EG_3 spends shorter runtime to obtain better congestion results. Finally, EG_{all} includes the post optimization stage to further improve routing results, so EG_{all} obtains better results than EG_3.

TABLE 3.6 ROUTE IND11 WITH DIFFERENT OBJECTIVES

	WCI (10^3)		ACE					
	100%	90%	0.5%	1%	2%	5%	10%	20%
OV	0.5	148.5	99.9	99.6	99.2	98.2	96.3	90.5
CO	4.5	50.2	99.7	97.8	95.3	90.8	87.0	83.2

3.4.2.2 Minimizing Overflow v.s. Minimizing Congestion

Compared to minimizing overflows, minimizing congestion ratio to approach TCR, say 80%, offers more useful congestion information to help designers to identify the hotspots' locations. Figures 3.7(a) and (b) show the congestion maps of design Ind11 obtained by EG_{all} addressing on overflow minimization and congestion ratio minimization, respectively. Obviously, Fig. 3.7(b) is more helpful to distinguish the hotspots' locations than Fig. 3.7(a). In Table 3.6, rows **OV** and **CO** list the congestion analysis for Figs. 3.7(a) and (b), respectively. Notably, EG_{all} both executes around 45 seconds to get the results in rows OV and CO, but OV has lower WCI(100) than CO because OV focuses on eliminating overflows.

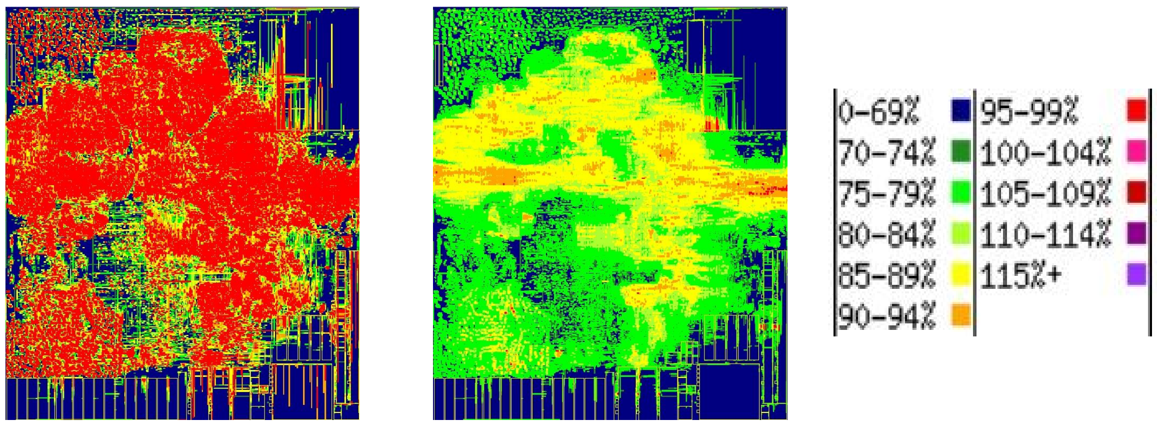


Fig. 3.7. Routing results of Ind11. (a) Minimizing overflows; (b) minimizing congestion ratio to approach 80%; (c) color scheme

3.4.2.3 Effectiveness of Throughput Controlling

Table 3.7 shows EG$_{\text{all}}$ with different values of t_u to trade off routing quality and runtime. Table 3.7 shows that the ACE metric monotonically improves and the runtime of EG$_{\text{all}}$ almost linearly increases when t_u increases. Also, WCI metric of the routing results except for design Ind9 monotonically improves when t_u increases. Table 3.7 reveals that the throughput controlling method can practically trade off the runtime and quality.

To demonstrate that EG$_{\text{all}}$ with the same value of t_u offers the stable runtime for any congestion circumstance, we add extra blockages to each g-cell in Ind11 and then use EG$_{\text{all}}$ with t_u=5 to route Ind11 with extra blockages. In Table 3.8, rows **Ex0**, **Ex2** and **Ex5** respectively show the congestion analyses for the routing results of Ind11 with 0%, 2% and 5% extra blockages, in which the runtimes of Ex0. Ex2 and Ex5 are respectively 45.6, 44.2 and 47.4 seconds. Table 3.8 indicates that the runtime of EG$_{\text{all}}$ is stable for different congestion circumstances, which can avoid the runtime of EG$_{\text{all}}$ being crazy when other tools that cooperate with EG$_{\text{all}}$ worsen the congestion largely. Figure 3.8 shows the congestion maps of Ind11 with 0%, 2% and 5% extra blockages.

TABLE 3.7 EFFECTIVENESS OF USING THROUGHPUT CONTROLLING METHOD TO TRADE OFF RUNTIME AND ROUTING QUALITY

	EG_{all}	CPU(s)	WCI(100)	WCI(90)	ACE(0.5)	ACE(1)	ACE(2)	ACE(5)	ACE(10)	ACE(20)	WL(10^7)
Ind7	$t_u = 3$	39.56	4209	29202	97.54	94.84	91.76	86.25	82.93	79.64	11.00
	$t_u = 5$	45.63	3257	24216	96.61	93.52	90.25	85.03	82.30	79.29	11.06
	$t_u = 7$	52.88	2941	18437	96.02	92.68	89.25	84.18	81.84	79.00	11.11
	$t_u = 9$	60.42	2751	14633	95.58	92.08	88.44	83.69	81.58	78.84	11.15
Ind8	$t_u = 3$	39.31	5807	23629	98.59	95.94	92.04	85.42	81.85	76.32	10.79
	$t_u = 5$	44.54	5000	18985	97.93	94.75	90.69	84.61	81.42	76.14	10.83
	$t_u = 7$	51.39	4834	16185	97.71	94.32	89.89	84.13	81.14	75.99	10.87
	$t_u = 9$	58.68	4705	14280	97.63	94.09	89.32	83.83	80.94	75.88	10.90
Ind9	$t_u = 3$	31.43	1142	12121	97.48	94.46	90.60	84.60	81.61	77.07	10.65
	$t_u = 5$	32.29	1098	9383	97.31	93.76	89.24	83.87	81.22	76.87	10.67
	$t_u = 7$	33.08	1004	6774	97.01	92.89	88.12	83.35	80.92	76.71	10.69
	$t_u = 9$	33.65	1021	4227	96.63	91.96	87.04	82.86	80.62	76.55	10.71
Ind10	$t_u = 3$	70.38	2608	36888	95.31	92.30	88.84	84.30	81.95	79.72	24.01
	$t_u = 5$	85.04	1681	13589	91.88	88.60	85.80	82.66	81.10	79.34	24.13
	$t_u = 7$	100.08	1411	10996	90.21	86.54	83.98	81.60	80.51	79.09	24.26
	$t_u = 9$	107.93	1395	10130	89.70	86.06	83.40	81.33	80.36	79.03	24.30
Ind11	$t_u = 3$	41.30	84	80837	94.68	93.31	91.92	89.90	86.95	83.48	14.72
	$t_u = 5$	45.55	77	77080	93.82	92.41	91.32	89.23	86.56	83.29	14.78
	$t_u = 7$	50.36	72	67575	93.32	92.07	91.06	88.69	86.18	83.09	14.82
	$t_u = 9$	55.95	72	59844	92.64	91.54	90.71	88.19	85.85	82.93	14.89
Ind12	$t_u = 3$	29.83	5260	52432	99.81	98.21	95.76	91.29	87.47	83.52	12.56
	$t_u = 5$	30.49	4480	50166	99.74	97.84	95.26	90.83	86.99	83.24	12.59
	$t_u = 7$	33.63	2716	47227	99.45	97.32	94.59	90.28	86.61	83.04	12.62
	$t_u = 9$	36.48	2388	43859	99.38	97.11	94.16	89.85	86.27	82.86	12.64
Ratio	$t_u = 3$	1	1	1	1	1	1	1	1	1	1
	$t_u = 5$	1.108	0.835	0.781	0.989	0.985	0.985	0.989	0.994	0.997	1.004
	$t_u = 7$	1.244	0.721	0.652	0.983	0.976	0.974	0.982	0.989	0.994	1.007
	$t_u = 9$	1.367	0.701	0.551	0.979	0.971	0.967	0.977	0.986	0.992	1.010

TABLE 3.8 ROUTE IND11 WITH EXTRA BLOCKAGES

	WCI (10^3)		ACE					
	100%	90%	0.5%	1%	2%	5%	10%	20%
Ex0	0.08	77.0	93.8	92.4	91.3	89.2	86.6	83.3
Ex2	0.13	87.1	95.8	94.2	93.2	91.1	88.3	85.2
Ex5	2.5	137.7	98.4	97.2	96.1	94.0	91.3	87.6

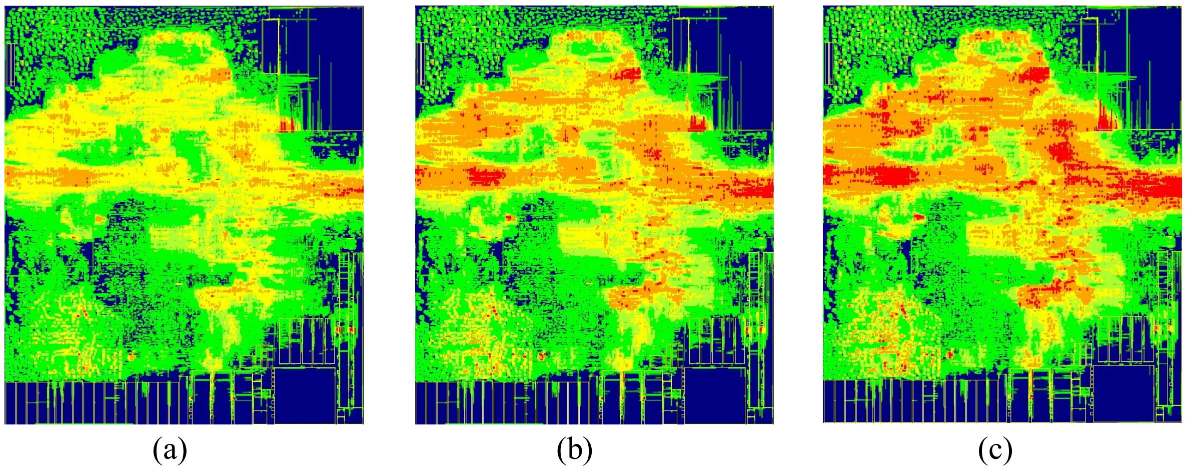


Fig. 3.8. Routing results of Ind11 (a) without extra blockages; (b) with 2% extra blockages; (c) with 5% extra blockages.

3.5 Summary

This work enhances Grace to fulfill the industrial requirements to develop a industrial RCE tool called E-Grace that takes TCR, local congestion, scenic and layer directive constraints into account. The goal of E-Grace is to minimize congestion ratio rather than overflows, the proposed TCR-driven R&R scheme can offer a satisfactory routing quality in a limited runtime budget. Further, a relaxation-legalization method is presented to handle the scenic constraint, which can escape from the local optimum to get better solution quality. Finally, by the proposed throughput controlling method, users can trade off the runtime and quality of E-Grace to get deterministic routing results. Experiments reveal that E-Grace is faster and more accurate than another industrial global-routing-based RCE.

Chapter 4 Ropt: Optimization of Placement Solutions for Routability

4.1 Introduction

To address the routability issue, the importance of the cooperation between placers and routing congestion estimators (RCE) is highlighted in this dissertation. Practically, the proposed RCE tool E-Grace has been adopted in industrial flow to cooperate with industrial placers for refining the placement solution's routability. Also, the academic routability-driven placers in [42-45] adopt global-routing-based RCEs to obtain a routing congestion map, and then move cells to optimize the congestion based on the map as well as routability. However, as cells move, the congestion map also changes, thereby affecting the effectiveness to improve routability.

To maintain the accuracy of the congestion map, IPR in [43] adopts global router FastRoute 2.0, developed in [14], to obtain a congestion map at the beginning of the detailed placement stage, and then incrementally updates the congestion map as cells move. For each cell c_i, IPR identifies the optimal region of c_i that minimizes the half-perimeter wirelength (HPWL) associated with c_i and swaps c_i with a cell in the optimal region. The cell chosen for swapping is the one that minimizes the congestion the most. After that, IPR reroutes the nets connected to the swapped cells. This method always maintains a global routing instance based on the current placement to offer accurate congestion information. However, IPR decides the cell locations according to its HPWL optimal region. If the optimal region is in a congested area, the congestion is hardly reduced.

To further compound the problem, most global routers [3-21] ignore the local congestion. As a consequence, placers that are guided by these routers may produce hard-to-route placement solutions in terms of detailed routing. In most global routing models, the placement is typically partitioned into an array of uniform G-cells, and global routers identify the G-cell-to-G-cell routes for each global net, a net whose terminals reside in different G-cells. However, the effects of the local nets, each of which has terminals residing within a G-cell, are typically ignored. To minimize local congestion, an estimation metric for local-routability was used by the detailed placer in [56] to guide the iterative movements of cells from high-cost G-cells to low-cost G-cells. A G-cell with a lower cost means that the routing within this G-cell is easier. However, the experiment in [56] reveals that this method may increase global congestion. To simultaneously consider global and local congestions, the authors of [53] took local pin density into account during global routing. If a G-cell has high pin density, the routing capacities from the G-cell to its neighboring G-cells are reduced, in order to avoid routing too many global paths through these G-cells. The remaining routing resources in the G-cell can be used for local routing.

ISPD 2011 and DAC 2012 routability-driven placement contests [46, 47] adopt NCTU-GR 2.0 [18] and BFG-R [7] to evaluate the routability of the contestants' placement solutions. However, the evaluation may be biased because NCTU-GR 2.0 and BFG-R do not consider local congestions. Thus, this work develops a translator to transform the placement solutions of the top four placers [38, 39, 42, 57] in DAC 2012 contest to the format that can be read in by commercial router Wroute [23] (version 3.0.61) to yield detailed routing results, and then analyzes how to optimize placement solutions that can reduce the routing violations in the detailed routing results.

Based on the observation for the detailed routing results of the placement solutions of [38, 39, 42, 57], this work develops a routability optimizer **Ropt** that takes a placement solution and optimizes its routability. Ropt combines both placement and global routing, and always maintains a global routing instance based on the current placement solution. The global routing instance is built on a local-routability-aware model. Therefore, the global routing instance provides both global and local congestion information to guide the placement algorithms. In contrast to CRISP [44] and CROP [58], which locally spread and shift cells, Ropt globally re-places cells to significantly improve the routability. Ropt has following innovations:

(a) We propose new capacity and demand estimations in the global routing model to account for both global and local congestion levels simultaneously, and generates a global routing instance based on the proposed model. Similar to [53], the proposed model also considers pin density. However, in contrast to [53], our model also places emphasis on the effect of blockages on routability.

(b) Rather than minimizing HPWL, the proposed routing-cost-driven global re-placement directly improves routability by minimizing the routing cost of nets, as the routing cost is defined in terms of congestion and wirelength. The estimation of routing cost is based on the global routing instance, which is updated after every cell movement.

(c) This work presents a legalization scheme to preserve the global routing instance after legalization. After that, the proposed local detailed placement further minimizes the local congestion and wirelength without increasing global congestion.

(d) In addition to using NCTU-GR 2.0 to evaluate the routability of the placement solutions, this work also uses Wroute [59] to obtain detailed routing results of the optimized placement solutions for the evaluation of routability.

The rest of this chapter is organized as follows. Section 2 introduces the problem of routability-driven placement. The case study for the placement solutions of [38, 39, 42, 57] is discussed in section 3. Section 4 presents the design flow of Ropt. Section 5 summarizes the experimental results. Conclusions are finally drawn in Section 6.

4.2 Problem Description

A circuit can be defined as a set of nets, where each net comprises a set of pins, each of which can be located on a cell, a macro, or a pad. The placement problem deals with the determination of the positions of cells and macros, with the goal of minimizing the (routed) wirelength used to connects all the nets, subject to the constraints that there are no overlaps between cells and macros. The placement area is partitioned into rows, the cells and macros must be aligned with the rows.

For the ISPD11 and DAC12 benchmark circuits [46, 47] used in this work, the positions of macros are fixed. Therefore, the objective of this work is to re-place movable cells to improve routability, subject to the constraints that the cells are aligned with the rows and that there are no overlaps in the final placement solutions. This work uses NCTU-GR 2.0 and Wroute to evaluate the routability of the placement solutions produced by the proposed optimizer Ropt on the circuits in [46, 47]. Global router NCTU-GR 2.0 is selected to evaluate the routability of placement solutions in the DAC12 and ICCAD12 placement contest. However, NCTU-GR 2.0 does not route local nets. Therefore, the impact of local congestion on the real routability cannot be evaluated properly. Indeed, a good global routing solution does not imply that a feasible detailed routing solution exists. For a thorough evaluation of the real routability of the placement solutions produced by Ropt, we also feed the placement solutions to Wroute to obtain the detailed routing results.

4.3 Case Study for Placement Solutions in DAC Contest

By holding ISPD 2011, DAC 2012 and ICCAD 2012 routability-driven placement contests, many researchers were attracted to develop effective routability-driven placers. The top four placers in DAC12 placement contests are, in alphabetical order, mPL [57], NTUplace [39], Ripple [38], and SimPLR [42], this work adopts the their placement solutions in DAC12 contest to perform detailed routing by Wroute and then do case study on their detailed routing results.

DAC12 contest adopts a global-routing-based metric to evaluate the routability of contestants' placement solutions, thus the placers in [57, 38, 39, 42] treat this metric as the objective. In DAC12 contest, each placement solution P is global routed by BFG-R and NCTU-GR 2.0. For each global routing solution G_P, a global congestion metric $PWC(G_P)$ is computed by calculating the congestion cost of each grid edge e in G_P, and then adding up the average congestion costs of the top 0.5%, 1%, 2%, and 5% congested grid edges. The score of a placement solution, denoted as $S(P)$, in the DAC12 contest is then computed as follows:

$$S(P) = HPWL(P) \times (1 + 0.03 \times PWC(G_P)), \qquad (4.1)$$

where $HPWL(P)$ denotes the total half-perimeter wirelength of P. A lower $S(P)$ implies better routability. Although this global-routing-based metric take total wirelength, average congestion, and peak congestion into account, but no experiment can show how accurately this metric can estimate real routability of detailed routing. The next subsection introduces how to read in the placement solutions in DAC 2012 and then to evaluate their real routability.

4.3.1 Framework for Performing Detailed Routing

The benchmark suite used in DAC 2012 placement contest [47] omits many physical design rules. Thus, it is not suited to be the benchmarks of detailed routing. In this work, we develop a translator based on the 28nm technology node to translate the placement solutions of [47] to LEF/DEF files and then run commercial router Wroute [59] on these circuits to obtain detailed routing results. Because the details of the design rules of the 28nm technology node are not readily available, we consult with several senior engineers from various companies (vendors, design companies, and manufacturers) to set reasonable routing parameters in the translator, in which the minimum wire width and via size are set to 42 *nm* and 56×56 nm^2, respectively. The minimum spacing rule is set to 42 *nm*. Based on the default setting in the benchmark suites [47], metal layers 1-4, 5-7, and 8-9 have 1X, 2X, and 4X the minimum wire width, via size, and spacing, respectively.

In the benchmark suites of [47], some pins are not accessible because they are blocked by macros. Thus, the translator moves these pins to the top layer of the macros to make these pins accessible. In addition, some pads are located at the same positions, which would cause routing violations. Therefore, the translator only reserves the pad that appears first in the benchmark files and removes the others that are located at the same position. As only few pads are removed, the effect of such removals should be negligible. In practice, many small obstacles and cell routings occupy the bottom layer and they consume most of the routing resources there, leaving very little routing resource available for the detailed routing of other components in the circuit. As [47] does not provide such information in the bottom layer, the translator conservatively assumes that the bottom layer is not available for the detailed routing of the placement solutions. However, vias are allowed to connect to pins at the bottom layer if the routing resource is enough to insert vias.

After obtaining the translated LEF/DEF files, Wroute is invoked to perform (global and detailed) routing in two iterations. The first iteration is the default routing mode and the second iteration is the post routing mode. In the default mode, Wroute reports the number of unroutable nets after the global routing stage and the unroutable nets are not routed in the detailed routing stage of the first iteration. In the post routing mode, Wroute routes *every* net in detail and further minimizes the number of violations, the total wirelength, and the number of vias of the routing result generated in previous iteration. The runtime limitation for each iteration is 24 hours. We have tried to run Wroute for more iterations to further optimize the detailed routing results. However, we typically observe only slight improvement over the solutions obtained using only two iterations. We conclude that for the purpose of evaluating the routability of placement solutions, running two iterations of Wroute strikes a good balance between routing quality and runtime.

4.3.2 Mismatch between Global and Detailed Routability

Table 4.1 compares the placement solutions of mPL, NTUplace, Ripple, and SimPLR based on the DAC12 metric, number of violations in detailed routing results, and TLMW, respectively. The definition of TLMW will be introduced later. In DAC12 contest, global routers BFG-R and NCTU-GR 2.0 are used to evaluate placement solutions. Thus, we use BFG-R and NCTU-GR 2.0 to get two global routing results G_1 and G_2 for each placement solution, and then compute congestion penalty $PWC(G_P)$ by averaging $PWC(G_1)$ and $PWC(G_2)$. Finally, the routability score for each placement solution is obtained via Eq. (4.1), which is shown in the first major column in Table 4.1. To evaluate which placer's solutions have better routability, we follow the convention used in the DAC12 contest and rank the placement solutions of each benchmark from 1 (best) to 4 (worst). After that, we average the rankings of the solutions in each column to get the average ranking (AR) for each placer. In DAC12 contest, a placer with smaller average ranking implies that this placer produces better routability placement solutions.

In Table 4.1, NTUplace has a much lower AR than other placers based on the DAC12 metric, which implies that NTUplace reduces both HPWL and global congestion well to obtain better routability scores $S(P)$ (Eq. (4.1)). However, based on the number of routing violations, NTUplace has a similar AR compared to other placers. We suspect that local congestion is the main cause of such behavior. To estimate the local congestion for each placement solution, we decompose every net into two-pin nets based on the topology of a RMST and then add up the Manhattan distance between the terminals of each local two-pin net to get the total local Manhattan wirelength (TLMW). If both terminals of a two-pin net are within a G-cell, the two-pin net is regarded as a local two-pin net. The right major column in Table 4.1 shows the TMLW of each placement solution, as well as the AR for each placer based on TMLW; a solution with shorter TMLW gets a better ranking. Table 4.1 reveals that the orders of AR based on the DAC12 metric and TLMW are reversed. This implies that a placement solution that has a better routability score evaluated by the DAC12 metric has longer TLMW. However, a longer TLMD may lead to local congestion and higher number of violations, which would cancel the benefit of performing well on the DAC12 metric. Thus, the four placers have similar ARs based on the number of violations.

TABLE 4.1 COMPARING THE PLACEMENT SOLUTIONS IN DAC12 CONTEST BASED ON THE DAC12 METRIC, VIOLATIONS AND TLMW

	DAC12 metric (10^7)				Violations				TLMW (10^5)			
	Ripple	mPL	SimPLR	NTUplace	Ripple	mPL	SimPLR	NTUplace	Ripple	mPL	SimPLR	NTUplace
s2	78.21	115.5	87.67	64.30	81227	113991	876	725813	97.14	79.33	98.99	99.48
s3	44.28	46.08	40.75	37.60	243	231	194	988	103.83	84.29	98.65	100.52
s6	36.93	41.12	36.94	36.68	232	231	361	637	118.60	93.39	115.40	117.11
s7	47.00	51.86	131.04	40.52	300	170	5402	168	171.10	144.15	168.27	175.28
s9	30.00	33.80	28.93	26.70	8136	68	22	89	102.69	84.97	103.12	103.18
s11	36.27	44.66	38.51	34.73	433	11009	1840	697	100.53	79.92	102.66	102.69
s12	37.38	53.03	37.69	31.68	155	343637	241	94	178.82	160.60	87.70	177.62
s14	23.89	27.45	25.68	22.96	19086	271799	224239	18446	65.39	51.01	63.25	66.86
s16	27.23	30.7	35.81	28.27	38	36	135	24	87.70	70.53	87.70	88.83
s19	16.95	22.78	16.63	15.33	110	276	72777	15114	56.64	47.30	56.58	58.96
AR	**2.2**	**3.8**	**2.9**	**1.1**	**2.3**	**2.6**	**2.7**	**2.4**	**3**	**1.1**	**2.2**	**3.7**

Figure 4.1 shows the placement solutions of s19 obtained by each placer. In these figures, the white crosses and green rectangles denote the routing violations and big macros, respectively. Recall

that the big macros are treated as routing blockages in the routing stage. We can see that most routing violations occur near blockages. Accordingly, we recognize some patterns that easily cause violations between blockages and cells. The placement solutions obtained by Ripple in DAC12 contest contain fewer such patterns, and therefore have fewer violations. These patterns will be presented in the next sub-section.

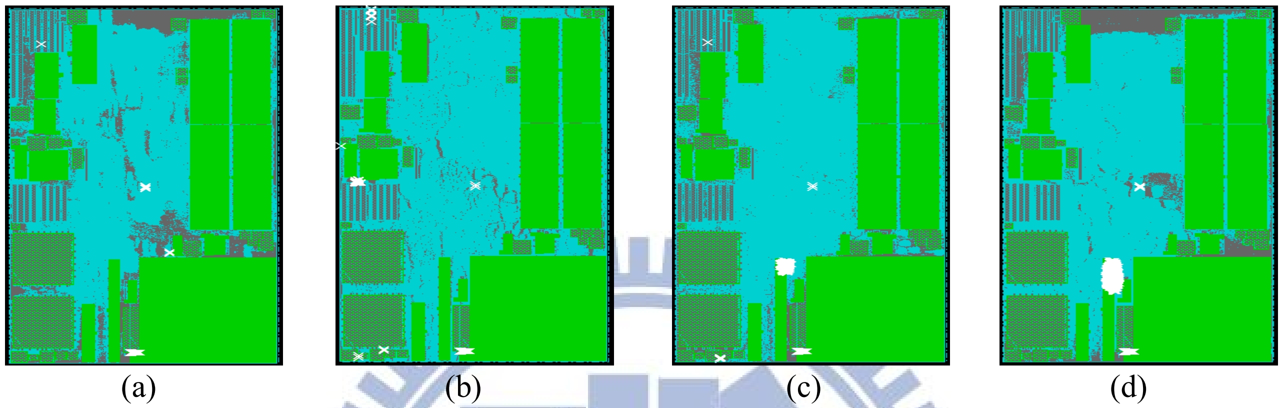
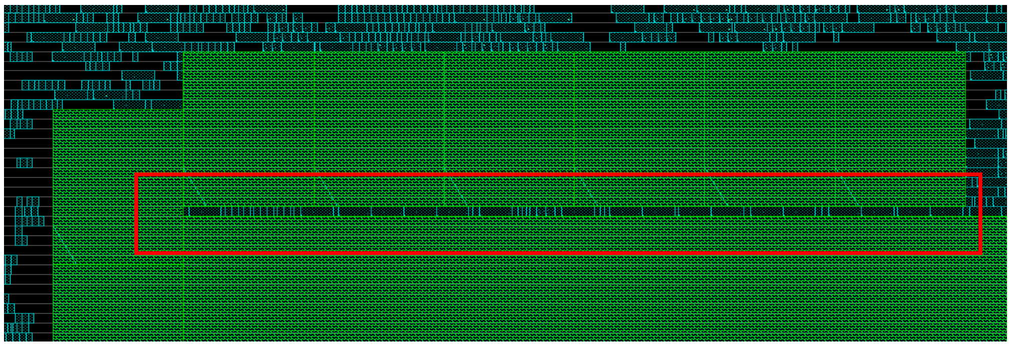


Fig. 4.1. Placement solutions of s19 obtained by (a) Ripple; (b) mPL; (c) SimPLR; (d) NTUplace.

4.3.3 What Causes Routing Violations

As placers are now getting better in resolving global congestion issue, it is more critical than ever that they consider local congestion in the optimization process. Otherwise, the placers may further accentuate the local congestion problem by turning global nets into local nets. To consider local congestion in the early stage, the works in [50, 56] developed local congestion estimation metrics based on the pin density; however, they did not consider the effect of blockages. Figure 4.1 shows that most violations occur near blockages. This implies that incorporating the effect of blockages into local congestion estimation metrics may improve the accuracy of the metrics in predicting local congestion.

We list some reasons why most violations near blockages based on our observations. (1) The global placement stage in placers would place cells on blockages. After legalization, these cells are pushed to the fringe of blockages, resulting in high congestion around the fringe of these blockages. (2) Routing wires cannot access pins that are too close to blockages because of spacing rules. (3) Although routing wires can stride over blockages via the high routing layers, the routing wires hardly use the high layers in the congested regions. (4) Routing wire bypassing blockages would increase the via count, which may make designs harder to route. However, the placement and global routing stages usually ignore the routing overhead incurred by vias.


(a)

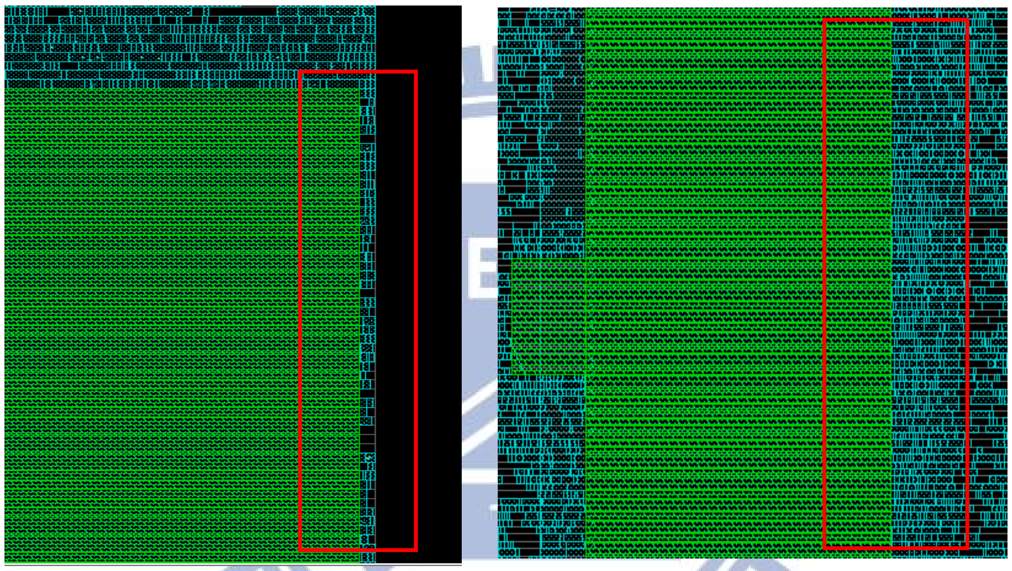

(b) (c)

Fig. 4.2. The local views of the most congested region in (a) the placement solution of mPL; (b) the placement solution of Ripple; (c) the placement solution of NTUplace for s19.

To take a deeper look why blockages cause violations, Fig. 4.2 provides the local views of the most congested regions in some placement solutions. In Fig. 4.2(a), many cells are placed in a narrow channel between blockages. In Fig. 4.2(b), many cells are placed into a narrow channel between a blockage and the design's boundary. In Fig. 4.2(c), many cells are adjacent to a big blockage. Note that we do not show the violations in Fig. 4.2 because doing so would make it impossible to view the cells and blockages, as the number of violations are just too high. Such patterns in Fig. 4.2, which cause violations, are commonly found in the placement solutions in DAC12 contests. Even if a placement solution can be easily globally routed, routing violations still exist in the placement solution if the patterns shown in Fig. 4.2 exist. We believe that violations can be reduced if placers avoid placing too many cells into narrow channels and preserve a small space between cells and big blockages.

4.4 Proposed Routability Optmizer

Figure 4.3 shows the design flow of Ropt. Given a set of nets and an initial placement, Ropt first constructs a global routing model with local routability taken into consideration. Next, each net is decomposed into two-pin nets based on the topology of a RMST, and then an initial global routing instance is generated by pattern routing and monotonic routing. Thereafter, routing-cost-driven global re-placement stage re-places each cell and immediately updates the global routing instance after each cell moves. Legalization stage aligns the cells on the row and removes overlaps; the goal of this stage is to preserve the global routing instance after legalization. Finally, local detailed placement further minimizes the local congestion and wirelength. Ropt iteratively refines the placement until a user defined terminating condition is met. In our implementation, we iterate the process up to two times (see Table 4.3 in Section 5 on the experimental setup).

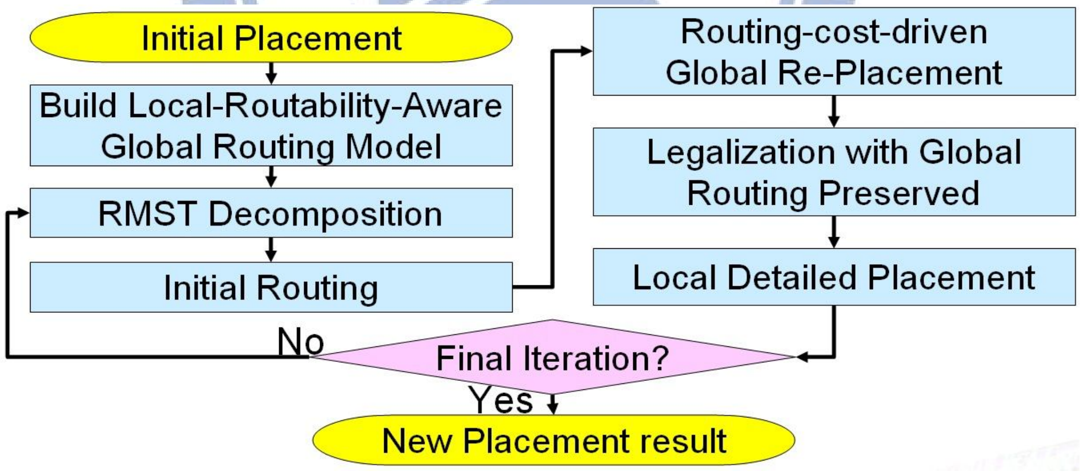

Initial Placement
Build Local-Routability-Aware Global Routing Model
RMST Decomposition
Initial Routing
Routing-cost-driven Global Re-Placement
Legalization with Global Routing Preserved
Local Detailed Placement
No
Final Iteration?
Yes
New Placement result

Fig. 4.3. Design flow of the proposed routability optimizer Ropt

4.4.1 Local-Routability-Aware Global Routing Model

Two global routing approaches are generally used in multi-layer global routing problem. The first one performs global routing on a three-dimensional (3D) grid graph to get a 3D routing result. The second approach first projects a 3D grid graph into a two-dimensional (2D) grid graph. Then, a 2D global router is used to obtain a 2D routing result. Finally, a layer assignment step transforms the 2D routing result to a 3D routing result. The former has a larger solution space than the latter has, but the latter is much faster than the former. In order to quickly extract the congestion information to guide the placement process, this work builds the global routing instance based on a 2D routing approach.

In the 2D global routing model, the given placement is modeled to a 2D grid graph $G(V, E)$, where V denotes the set of G-cells, and E refers to the set of grid edges. Each grid edge connects two adjacent G-cells. The capacity $c(e)$ of grid edge e is the number of routing tracks that can legally cross the abutting boundary, and the demand $d(e)$ of e is the number of global routing path passing through e. If $d(e)$ exceeds $c(e)$, too many global routing paths are passing through e, which implies global congestion. Thus, global congestion can be measured in terms of $d(e)$ and $c(e)$. Note that, two terms G-cell and cell are used in this chapter; a G-cell means a bin in the global routing model, while a cell means a primitive instance to be placed in the placement problem.

This work accounts for local-routability in the estimations of capacities and demands of grid edges. Therefore, when the global router in the proposed flow finds the least-cost routing paths whose cost formulation is based on the new capacity and demand estimations, the global and local congestion can be simultaneously considered and reduced. The following equation has been used in many 2D global routers to estimate grid edge capacity [7-21]:

$$c(e) = \sum_{l \in L} (T_l(b_1, b_2) - BT_l(b_1, b_2)), \qquad (4.2)$$

where b_1 and b_2 are the adjacent G-cells of grid edge e, L is the set of layers, and $T_l(b_1, b_2)$ and $BT_l(b_1, b_2)$ represent the number of routing tracks and the number of routing tracks blocked by big macros at layer l between b_1 and b_2, respectively. This capacity estimation disregards the vias and local routes in the adjacent G-cells. However, the presence of vias and local routes can make it hard for a design to be detailed routed. Based on the case study in previous section, we observed that the vias and local routes in b_1 and b_2 increase as the number of blocked tracks between b_1 and b_2 increases. The increased vias and local routes would further block the tracks between b_1 and b_2, and more blocked tracks necessitate more detour and vias for a path to cross the boundary between b_1 and b_2. As the number of blocked tracks increases, the edge capacity decreases more than linearly. To capture this effect, this work estimates the capacity of a grid edge with the following equation:

$$c'(e) = \left\lceil (1 - \frac{\sum_{l \in L} BT_l(b_1, b_2)}{\sum_{l \in L} T_l(b_1, b_2)})^{\alpha} \times \sum_{l \in L} T_l(b_1, b_2) \right\rceil, \tag{4.3}$$

where α is a user defined constant; we set α to be 2.5 in this work. (In this work, there are many parameters that can be tuned. We select these parameters based on empirical results. However, once set, these parameters stay the same throughout the experiment. We do not change the parameters according to the benchmark circuits, although it may be wise to do so to obtain better results.) Moreover, it has been demonstrated in [50] that the pin densities in b_1 and b_2 also affect the routability between b_1 and b_2. Thus, we estimate the demand placed on e as follows:

$$d'(e) = d(e) + \left\lceil \beta \times (\frac{p(b_1)}{\sum_{l \in L} A_l(b_1)} + \frac{p(b_2)}{\sum_{l \in L} A_l(b_2)}) \right\rceil, \tag{4.4}$$

where $d'(e)$ denotes the number of routing tracks used by global and local paths between b_1 and b_2, in which $d(e)$ is the number of routing tracks used by global paths, and the second term denotes the number of estimated routing tracks used by local paths. Since detailed routing is yet to be performed to realize local paths, we simply use pin density to estimate the number of local paths, the concept is similar to [50]. In Eq. (4.4), $p(b_1)$ and $p(b_2)$ denote the number of pins in b_1 and b_2, respectively. $A_l(b_1)$ and $A_l(b_2)$ denote the areas which are not covered by macros in b_1 and b_2 at layer l, respectively. β is a user defined constant, and we set it to be 1500.

Many global routing papers [7, 14, 11, 17] discuss how to formulate the routing cost in terms of the global congestion based on $c(e)$ and $d(e)$ (and possibly other relevant metrics) and then find the least-cost routing path for each net to minimize the global congestion. This work adopts the routing cost formulation presented in [17], but replaces $c(e)$ and $d(e)$ by $c'(e)$ and $d'(e)$, respectively, in order to consider local-routability. This local-routability–aware routing cost function is used in the initial routing and global re-placement stages.

4.4.2 Routing-cost-driven Global Re-Placement

As minimizing the total routing cost can directly improve the congestion, we formulate such a routing cost minimization problem as that of finding the optimal placement G-cell b_m for a movable cell c_i in the global re-placement stage to minimize the routing cost. Also, to avoid placing too many cells into a G-cell, each G-cell has to satisfy a bin density constraint. The problem of Finding Optimal Placement G-cell (FOPG) for c_i is concisely formulated by the following equations (for ease of explanation, if c_i is placed into b_m, we assume all pins of c_i are inside b_m):

$$\min_{b_m \in R} \sum_{n(p_i,p_j)\in N(c_i)} \sum_{e \in r(b_m,b(p_j))} routC(e) \tag{4.5}$$

$$routC(e) = C_1 + C_2 \Big/ \left(1 + C_3^{C_4 \times (c'(e) - d'(e))}\right) \tag{4.6}$$

$$\text{s.t.} \qquad D(b_m) \le D_t \tag{4.7}$$

Equation (4.5) is the objective equation of FOPG problem, R is the set of candidate G-cells for placing c_i, $N(c_i)$ denotes the set of two-pin nets connecting to c_i, and $n(p_i, p_j)$ represents a two-pin net whose terminals are pins p_i and p_j. Pin p_i is on c_i and p_j is on the other cell, macro or pad. When c_i is placed into b_m, p_i is in b_m; $b(p_j)$ denotes the G-cell containing p_j. $r(b_m, b(p_j))$ denotes the least-cost global routing path from b_m to $b(p_j)$, e is a grid edge passed by $r(b_m, b(p_j))$, and $routC(e)$ denotes the routing cost of e. The formulation of $routC(e)$ in Eq. (4.6) is inspired by [17], in which C_1, C_2, C_3 and C_4 are user defined constants; we set them to 25, 20, 2.72 and 0.3, respectively. Moreover, Eq. (4.7) is the bin density constraint; $D(b_m)$ denotes the bin density of b_m, which is the total area of cells in b_m divided by the area of b_m. If $D(b_m)$ exceeds D_t, it is called bin overflow. In this work, we set D_t to 0.9.

The proposed routing-cost-driven global re-placement stage uses a heuristic method to solve the FOPG problem. This method first identifies a set of candidate G-cells for placing c_i and then evaluates the cost of placing c_i in each candidate G-cell. Finally, c_i is placed into the G-cell with minimal placing cost. The cost of placing c_i in b_m is calculated as follows:

$$p(b_m, c_i) = desC(b_m) \times h(b_m) + \sum_{n(p_i,p_j)\in N(c_i)} \sum_{e \in mr(b_m,b(p_j))} routC(e), \tag{4.8}$$

$$desC(b_m) = \begin{cases} \kappa \times D(b_m) & \text{if } D(b_m) \le D_t \\ \kappa \times D_t + \mu \times (D(b_m) - D_t) & \text{otherwise} \end{cases}, \tag{4.9}$$

where $p(b_m, c_i)$ denotes the cost of placing c_i in b_m, $desC(b_m)$ denotes the penalty because of the bin density of b_m, and $mr(b_m, b(p_j))$ denotes the least-cost monotonic routing path from b_m to $b(p_j)$. Moreover, $h(b_m)$ is the history cost of b_m; it has an initial cost of 1 and it increases by 1 when the placement of a cell in b_m causes bin overflow. Incrementally increasing history cost can gradually decrease bin overflows. In Eq. (4.9), κ and μ are user defined constants. In this work, we set κ and μ to 1 and 1000, respectively. Accordingly, if $D(b_m)$ exceeds D_t, the bin density cost becomes very large.

```
Algorithm Solving FOPG problem
Input: a movable cell $c_i$, global routing instant $O$
1. $R$←identifyPlacingRegion($c_i$)
2. Rip_up($c_i$, $N(c_i)$, $O$)
3. $Tc$←getBinDesityCost($R$)
4.     foreach two-pin net $n(p_i, p_j) \in N(c_i)$
5.         $Tr$←getRoutingCost($b(p_j)$, $R$, $O$)
6.         $Tc = Tc + Tr$
7.     end foreach
8. Place $c_i$ into the G-cell with the minimal placing cost
9. Monotonic_Routing($N(c_i)$ ,$O$)
```

Fig. 4.4. Pseudo code of the algorithm for FOPG problem.

Figure 4.4 shows the algorithm to solve the FOPG problem. Line 1 first identifies a minimum bounding box enclosing c_i and the pins with connection to c_i, and then extends the boundaries of the box by γ units of G-cells. The G-cells within the box are regarded as the candidate G-cells for placing c_i. For example, in Fig. 4.5(a), the G-cells in the red box are the candidate G-cells for placing c_i when γ is one. A larger γ offers a bigger solution space but longer runtime; γ is set to 5 in this work. In line 2, c_i is ripped-up from the placement and the two-pin nets in $N(c_i)$ are also ripped-up from the global routing instance O (Fig. 4.5(b)).

Line 3 computes the bin density and the associated penalty (via Eq. (4.9)) of each G-cell in R and stores the penalties in matrix Tc. In Line 5, monotonic routings from $b(p_j)$ to four corner G-cells in R are performed. Monotonic routing can obtain a least-cost monotonic path from the source to each G-cell in the minimum bounding box enclosing the source and target. So, the monotonic routings from $b(p_j)$ to four corner G-cells in R can obtain the least-cost of the monotonic path from $b(p_j)$ to every G-cell in R. After that, the routing costs are stored in matrix Tr. In Fig. 4.5(c), the gray rectangles represent congested regions, the arrows depict the directions of monotonic routings, and the number in each G-cell denotes the least-cost of the monotonic path from $b(p_1)$ to each G-cell. For simplicity, we assume

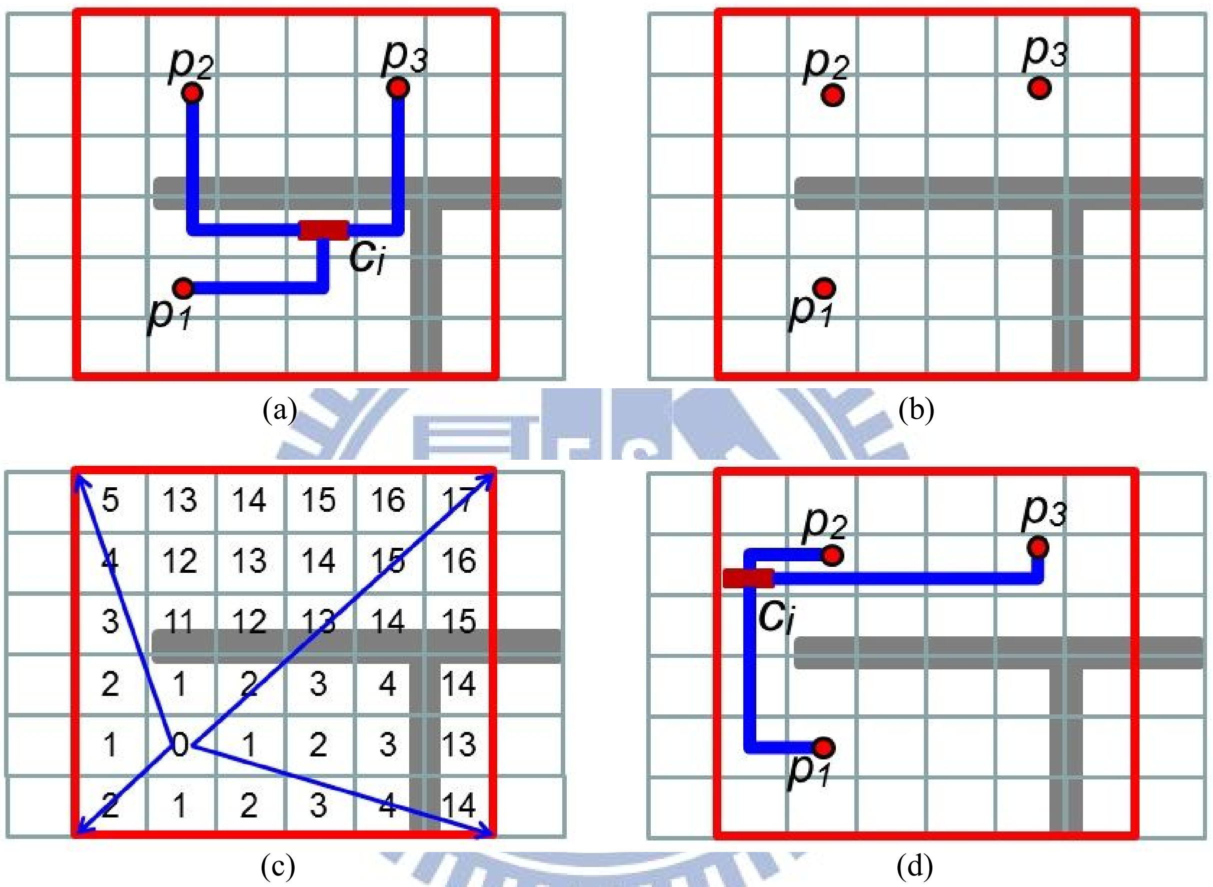


Fig. 4.5. (a) the G-cells in the red box are the placing candidates for c_i; (b) rip-up c_i and the two-pin nets in $N(c_i)$; (c) the number in each G-cell denotes the least-cost of monotonic path from $b(p_1)$ to each G-cell; (d) place c_i into the G-cell with minimal placing cost and reroute the two-pin nets in $N(c_i)$.

in this example that $routC(e)$ is 10 if e crosses a congested region, and 1 otherwise. When the iterations from lines 4 to 7 in Fig. 4.4 terminate, Tc contains for each G-cell in R, the cost of placing c_i in that G-cell. Finally, line 8 places c_i into the G-cell with minimal cost and line 9 routes the monotonic paths of every two-pin net in $N(c_i)$ and updates O (Fig. 4.5(d)). The time complexity of this algorithm is $O(|R|*|N(c_i)|)$ where $|R|$ and $|N(c_i)|$ denote the number of G-cells in R and the number of two-pin nets connected to c_i, respectively.

The global re-placement stage performs the algorithm in Fig. 4.4 for every cell in each round. In our implementation, this stage runs for four rounds. Notably, this stage simply places cells at the center of the assigned G-cells. The definite locations of cells are decided in the following stages: legalization and local detailed placement.

4.4.3 Legalization with Global Routing Preserved

After global re-placement, many cells have overlaps and are not aligned with the rows. The duty of legalization is to align cells on the rows and remove overlaps. Most legalizers [60, 61] minimize the total displacement of cells as the main objective is to ensure consistency between the global placement solution and the legalized solution. Instead, the objective of our legalizer is to preserve the global routing instance after legalization. In order to minimize the change to the global routing instance, if the global re-placement stage places a cell in G-cell b_m, our legalizer attempts to keep this cell in b_m.

Our legalizer is similar to Abacus [60], but with a different objective. First, every cell is sorted in increasing ordering according to its x-coordinate. Next, for each cell c_i, our legalizer tentatively moves c_i to its neighboring rows r_k and calculates the cost of moving c_i to r_k. Then, c_i is moved to the best row with the lowest cost. In Abacus, the cost formulation is based on the displacement of c_i. Instead, this work calculates the cost as follows,

$$\begin{aligned} mc(c_i,b_m,r_k) = {} & \mu \times ao(c_i,b_m,r_k) + \sigma \times \sum\nolimits_{p\in P(c_i)} d(b_m, b(p_k)) \\ & + m(c_i,r_k) + cu(r_k) \end{aligned}, \quad (4.10)$$

where c_i is placed into G-cell b_m in the global re-placement stage, $mc(c_i, b_m, r_k)$ denotes the cost of moving c_i to row r_k, $ao(c_i, b_m, r_k)$ denotes the area of c_i out of b_m when c_i is moved to r_k, $P(c_i)$ denotes the set of pins belonged to c_i, $b(p_k)$ denotes the G-cell containing p when c_i is moved to r_k, and $d(b_m, b(p_k))$ denotes the index distance between $b(p_k)$ and b_m. For example, if the indexes of b_m and $b(p_k)$ in the grid graph are (x, y) and (x', y'), respectively, $d(b_m, b(p_k))$ is $|x-x'|+|y-y'|$. Moreover, $m(c_i, r_k)$ is the displacement of c_i when c_i is moved to r_k, $cu(r_k)$ is the capacity utilization of r_k which is between zero and one. μ and σ are user defined constants. In our implementation, μ and σ are set to 1000 and 100, respectively. Since the global routing instance would change if c_i moves out of b_m, Eq. (4.10) gives the high penalty of $mc(c_i, b_m, r_k)$ if c_i moves out of b_m.

```
Algorithm    Local Detailed Placement
Input: Grid Graph $G(V, E)$
1.  for $it$ =1 to $u$
2.      foreach bin $b_m \in V$
3.          foreach cell $c_i \in C(b_m)$
4.              Cell_Swapping($c_i$, $C(b_m)$)
5.          foreach segment $s_j$ within $b_m$
6.              Sliding_Window($s_j$, $C(b_m)$)
7.          foreach cell $c_i \in C(b_m)$
8.              Moving_to_Empty_Spot($c_i$, $C(b_m)$, $b_m$)
9.      end foreach
10. end for
```

Fig. 4.6. Pseudo code of local detailed placement

4.4.4 Local Detailed Placement

This stage addresses the problem of minimizing the local wirelength in each G-cell to reduce local congestion. To avoid degrading the global routability, this stage does not move a cell from its original G-cell to another G-cell so as to ensure that the global routing instance remains the same. Performing detailed routing in each G-cell can get accurate local wirelength and congestion information but is time-consuming. Thus, this stage simply uses Manhattan distance to measure the local wirelength in each G-cell. If the terminals of a two-pin net n are both in G-cell b_m, the local wirelength of n is measured by the Manhattan distance between its two terminals. If a terminal of n is outside b_m, the terminal is projected on the boundary of b_m and the local wirelength of n in b_m is measured by the Manhattan distance between the projected terminal and inside terminal.

Figure 4.6 shows the pseudo code of local detailed placement, in which $C(b_m)$ denotes the set of cells that are entirely in b_m, i.e. the cells in $C(b_m)$ do not cross the boundaries of b_m; s_j denotes a segment of rows within b_m, u is a user defined terminating condition, we set it to be 5. Local detailed placement adopts cell swapping, sliding window and moving cells to empty spots methods to greedy improve local wirelength. In Fig. 4.6, line 4 tentatively swaps c_i with other cells in $C(b_m)$. If there exists at least a legal swap that can reduce the local wirelength, line 4 picks the legal tentative swap with the best improvement to perform an actual swap. A legal swap means that the swapped cells are entirely in b_m and they do not cause cell overlap. Line 6 adopts the sliding windows method [62] to re-order the cells in s_j for minimizing the local wirelength. We use a window size of 5 cells in this work. Line 8 tentatively moves c_i to an empty spot in b_m. If there exists at least a legal move that can reduce the local wirelength, line 8 picks the legal tentative move with the best improvement to perform a real move. Again, a legal move is one that does not cause cell overlap.

TABLE 4.2 BENCHMARKS' INFORMATION

Bench-mark	Total Nodes	Movable Cells	Fixed Macros	Fixed Pads	Total Nets	total Pins	Design Util.(%)	Design Den.(%)
s2	1014029	921273	59312	33444	990899	3228345	76	28
s3	919911	833370	55033	31508	898001	3110509	73	42
s6	1014209	919093	65316	29800	1006629	3401199	73	43
s7	1364958	1271887	66995	26076	1340418	4935083	76	58
s9	846678	789064	37574	20040	833808	2898853	73	47
s11	954686	859771	67303	27612	935731	3071940	79	40
s12	1293433	1278084	8953	6396	1293436	4774069	56	44
s14	634555	567840	44743	21972	619815	2049691	72	50
s16	698741	680450	419	17872	697458	2280931	69	46
s19	522775	506097	286	16392	511685	1714351	78	49

4.5 Experimental Results

The proposed algorithms are implemented in C/C++ on a quad-core 2.4 GHz Xeon-based linux server with a 50GB memory (only a single core is used). This work uses the placement solutions of NTUplace to be the input placement for Ropt. NTUplace won the DAC12 routability-driven placement contest [47], and the placement solutions of NTUplace in the contest are downloaded from [63]. Table 4.2 shows the design information in the benchmark suit used in the DAC12 placement contest, in which "Total Nodes" denote the total number of movable cells, fixed macros and fixed pads, "Design Util."

denotes the ratio of the total area of the movable cells and fixed macros to the area of the placement region, and "Design Den." denotes the ratio of the total area of the movable cells to the free-space in the design. The free-space is the area of the placement region minus the total area of the fixed macros.

This work proposes four features: local-routability-aware global routing model (LGM) with new resource and demand estimates, routing-cost-driven global re-placement (RGP), legalization with global routing preserved (LRP) and local detailed placement (LDP). To show the effectiveness of each feature, several versions of Ropt are built to evaluate the effects of these features. Table 4.3 lists the features of each version, in which the column "iteration" is the terminal condition in Fig. 4.3. Notably, the versions without LGM adopt $c(e)$ and $d(e)$ rather than $c'(e)$ and $d'(e)$ to formulate the routing cost in Eq. (4.6), and the Ropt_1, which does not use LRP, uses Abacus [60] instead of the proposed legalizer.

TABLE 4.3 ROPT WITH DIFFERENT FEATURES

Versions	Features				
	LGM	RGP	LRP	LDP	iteration
Ropt_1		✓			1
Ropt_2		✓	✓		1
Ropt_3		✓	✓		2
Ropt_4	✓	✓	✓		2
Ropt_5		✓	✓	✓	2
Ropt_6	✓	✓	✓	✓	2

4.5.1 Global Routability: Evaluation by NCTU-GR 2.0

For the evaluation of the effect of the proposed algorithm on global routability, we use Ropt_1, Ropt_2 and Ropt_3 to optimize the placement solutions of NTUplace, and then use NCTU-GR 2.0 to generate the global routing result of each placement solution. Because LGM and LDP address the issues of local wirelength and congestion, the effectiveness of Ropt_4, Ropt_5 and Ropt_6 cannot be evaluated by global routers. Therefore, Ropt_4, Ropt_5 and Ropt_6 are evaluated by Wroute in the next sub-section. Notably, NCTU-GR 2.0 is a public global routing tool with several tunable parameters. We set the parameters of via cost, wirelength optimization level, pattern routing iteration, monotonic routing iteration and post routing iteration in NCTU-GR 2.0 to 1, 50, 2, 2 and 1, respectively. The rip-up and rerouting stage in

NCTU-GR 2.0 iterates until either an overflow-free result is obtained or the iteration number is more than 25.

TABLE 4.4 GLOBAL ROUTING RESULT COMPARISON BETWEEN NTUPLACE, ROPT_1, ROPT_2 AND ROPT_3

	NTUplace			NTUplace+Ropt_1				NTUplace+Ropt_2				NTUplace+Ropt_3			
	WL_g	Via_g	R_{cpu}	WL_g	Via_g	R_{cpu}	P_{cpu}	WL_g	Via_g	R_{cpu}	P_{cpu}	WL_g	Via_g	R_{cpu}	P_{cpu}
s2	178.43	55.37	471.62	179.51	62.81	325.89	611.98	173.75	49.00	236.08	594.14	173.07	48.22	234.00	1143.15
s3	109.15	50.75	270.28	109.99	57.45	283.69	483.58	104.44	46.37	204.97	479.23	103.61	45.62	184.03	908.00
s6	104.37	51.64	169.73	107.09	61.26	175.29	445.01	100.67	47.11	120.18	445.15	100.09	46.34	115.86	829.15
s7	127.57	75.00	128.12	129.39	85.02	118.74	746.92	122.02	69.24	100.28	739.33	121.11	68.27	94.44	1409.29
s9	77.08	41.15	88.69	78.99	49.06	78.65	638.75	73.89	37.59	62.85	643.26	73.37	36.97	60.62	1152.00
s11	103.38	45.77	109.75	105.69	53.35	102.85	322.33	100.38	41.86	90.86	322.34	99.82	41.33	83.11	615.59
s12	109.52	70.80	111.74	113.16	83.85	108.24	1002.11	104.22	64.02	82.20	1048.66	103.20	62.97	81.92	1821.92
s14	69.19	33.95	87.11	70.85	38.72	90.25	307.41	67.63	31.20	72.07	320.11	67.31	30.82	66.02	576.04
s16	79.13	33.06	154.94	81.84	41.34	250.24	261.45	76.77	29.69	91.18	263.02	76.42	29.12	81.75	500.49
s19	46.79	25.19	41.38	48.02	30.40	39.03	608.24	44.68	22.46	32.30	612.08	44.42	22.11	31.33	1101.72
Ratio_{ind}	**1**	**1**	**1**	**1.022**	**1.173**	**1.009**		**0.964**	**0.908**	**0.722**		**0.958**	**0.894**	**0.681**	
Ratio_{sum}	**1**	**1**	**1**	**1.020**	**1.167**	**0.963**		**0.964**	**0.909**	**0.669**		**0.958**	**0.895**	**0.632**	

Table 4.4 shows the routing results of each placement solution, in which WL_g, Via_g, R_{cpu} and P_{cpu} denote the total global routing wirelength (10^5), global routing via count (10^5), NCTU-GR 2.0's runtime (sec) and Ropt's runtime (sec), respectively. The routing results of NTUplace are treated as the baseline; Ratio_{ind} is the average of the ratio of individual entries in the same column, while Ratio_{sum} denotes the ratio of the sum of each column. The routing overflows are not listed in Table 4.4 since every routing result is overflow-free. Table 4.4 reveals that Ropt_1 has worse WL_g, Via_g and R_{cpu} than NTUplace. This implies that even though the global re-placement may optimize a global routing instance, the legalization stage can worsen the global routing result when the legalizer is oblivious to the global routing instance. On the other hand, Ropt_2, which uses the proposed legalizer to preserve the global routing instance, can on the average improve WL_g, Via_g and R_{cpu} by 3.6%, 9.2% and 27.8%, respectively, when compared to NTUplace. In addition, the overflow issue in the Ropt_2's solutions can be easily resolved by the pattern and monotonic routing stages in NCTU-GR 2.0. Consequently,

NCTU-GR 2.0 does not have to invoke the more time-consuming maze routing stage as frequently. Therefore, the runtime of NCTU-GR 2.0 decreases. For example, in case s16, after the monotonic routing stage in NCTU-GR 2.0, 105430 and 14363 overflows remain in the placement solutions of NTUplace and Ropt$_2$, respectively. Also, Ropt$_2$ allows many nets to have simple routing solution, thereby reducing via count. Moreover, Ropt$_3$ can on the average reduce WL$_g$, Via$_g$ and R$_{cpu}$, respectively, by 4.2%, 10.6% and 31.9%, when compared to NTUplace. Table 4.4 shows that the placement solutions can be further improved as the Ropt runs more iterations, but the improvement gradually diminishes.

TABLE 4.5 DETAILED ROUTING RESULTS OF NTUPLACE

Bench marks	NTUplace				
	NUN	Vio	WL$_d$ (10^7)	Via$_d$ (10^6)	R$_{cpu}$
s2	2169	725813	67.80	12.28	48:33:08
s3	1450	988	39.99	11.05	13:06:50
s6	921	637	39.17	11.49	11:02:01
s7	419	168	48.31	16.93	13:17:31
s9	1112	89	29.19	9.62	08:49:18
s11	313	697	38.63	10.19	11:52:41
s12	120	94	42.73	17.01	11:26:41
s14	2352	18446	26.81	7.46	21:57:51
s16	78	24	29.19	7.72	06:59:38
s19	414	15114	18.11	5.99	16:42:50

4.5.2 Effective Routability: Evaluation by Wroute

Because global routers ignore the local nets within G-cells, using global routers to evaluate placement solutions may encourage placers to push many nets into a G-cell to improve global routing results. However, the local congestion would make it harder to route such designs in the detailed routing stage. To examine that Ropt can really improve routability, we via the proposed translator feed the placement solutions to Wroute, and then evaluate the routability based on their detailed routing results. Table 4.5 shows the detailed routing results of the placement solutions obtained by NTUplace, in which NUN is an indicator in Wroute to estimate the global routability (lower NUM means better global routability), Vio denotes the routing violations that are caused by opens, shorts or spacing errors, and R_{cpu} (hh:mm:ss) denotes the runtime of Wroute.

TABLE 4.6 DETAILED ROUTING RESULT COMPARISON BETWEEN NTUPLACE, ROPT$_3$, ROPT$_4$, ROPT$_5$ AND ROPT$_6$

	NTUplace+Ropt$_3$						NTUplace+Ropt$_4$					
	NUN	Vio	$WL_d(10^7)$	$Via_d(10^6)$	R_{cpu}	P_{cpu}	NUN	Vio	$WL_d(10^7)$	$Via_d(10^6)$	R_{cpu}	P_{cpu}
s2	1003	800	67.25	12.30	14:03:40	00:18:45	552	693	67.27	12.27	13:55:41	00:24:13
s3	728	369	39.62	11.40	11:34:13	00:14:58	270	205	39.57	11.37	10:31:41	00:17:28
s6	243	267	39.46	12.03	11:27:09	00:14:14	113	217	39.36	12.00	10:38:46	00:16:27
s7	192	152	48.11	17.34	12:44:58	00:24:46	110	132	48.06	17.33	12:05:44	00:27:17
s9	394	875	29.44	9.97	10:11:47	00:20:38	51	37	29.42	9.95	07:24:35	00:22:59
s11	258	464	38.80	10.50	11:30:38	00:11:17	119	421	38.85	10.50	10:22:15	00:12:52
s12	119	1226	43.20	18.17	16:14:58	00:33:23	65	431	42.94	18.07	14:41:42	00:35:32
s14	2007	15736	27.18	7.78	23:19:04	00:10:40	1482	22656	27.09	7.72	19:27:56	00:11:46
s16	65	26	29.70	8.22	06:45:03	00:08:40	0	22	29.66	8.23	06:38:53	00:10:13
s19	386	6814	18.30	6.31	16:10:48	00:19:23	287	2411	18.20	6.29	11:49:00	00:21:31
Ratio$_{ind}$	**0.648**	**2.763**	**1.005**	**1.040**	**0.971**		**0.312**	**0.924**	**1.003**	**1.037**	**0.851**	
Ratio$_{sum}$	**0.577**	**0.035**	**1.003**	**1.039**	**0.818**		**0.326**	**0.036**	**1.001**	**1.036**	**0.718**	

	NTUplace+Ropt$_5$						NTUplace+Ropt$_6$					
	NUN	Vio	$WL_d(10^7)$	$Via_d(10^6)$	R_{cpu}	P_{cpu}	NUN	Vio	$WL_d(10^7)$	$Via_d(10^6)$	R_{cpu}	P_{cpu}
s2	989	693	66.55	11.96	14:26:49	00:23:58	538	664	66.59	11.94	12:49:11	00:27:01
s3	771	312	39.00	11.00	10:41:40	00:19:37	279	201	38.95	10.95	09:10:16	00:20:01
s6	267	283	38.79	11.60	09:59:04	00:18:55	108	218	38.71	11.56	09:38:43	00:19:37
s7	191	174	47.42	16.94	11:46:12	00:29:23	113	136	47.38	16.92	12:20:41	00:31:32
s9	357	69	28.91	9.67	07:40:39	00:25:57	58	24	28.91	9.65	07:48:57	00:26:16
s11	234	441	38.27	10.23	10:35:36	00:14:30	147	422	38.32	10.22	10:53:57	00:15:07
s12	103	259	42.11	17.41	12:11:41	00:42:11	72	96	41.83	17.32	10:54:10	00:42:57
s14	1998	14913	26.82	7.54	22:40:07	00:12:35	1457	10374	26.74	7.48	17:22:44	00:13:53
s16	62	26	29.15	7.84	06:15:12	00:11:25	0	24	29.13	7.86	06:10:00	00:13:03
s19	367	2745	17.93	6.07	10:01:30	00:23:06	268	265	17.83	6.05	06:40:36	00:23:51
Ratio$_{ind}$	**0.619**	**0.803**	**0.988**	**1.005**	**0.826**		**0.322**	**0.483**	**0.987**	**1.002**	**0.759**	
Ratio$_{sum}$	**0.571**	**0.026**	**0.987**	**1.005**	**0.710**		**0.325**	**0.016**	**0.985**	**1.002**	**0.634**	

Table 4.6 compares the detailed routing results of Ropt$_3$, Ropt$_4$, Ropt$_5$ and Ropt$_6$ with NTUplace. Because Ropt$_3$ improves the global routing results, NUN is reduced by 35.2% on the average. However,

Ropt_3 does not consider the local wirelength and congestion, resulting in increases in violations, wirelength and vias. Furthermore, Ropt_4 uses LGM to consider global and local congestions simultaneously. Therefore, Ropt_4 yields fewer NUN and violations than Ropt_3. Because Ropt_4 does not minimize the local wirelength, local routing can still cause violations, see s14 for example. To minimize the local wirelength and congestion, Ropt_5 uses LDP to get fewer violations, shorter wirelength, and fewer vias than Ropt_3. Notably, since the only difference between Ropt_5 and Ropt_3 is LDP, the global routing results of Ropt_5 and Ropt_3 are similar. Thus, Ropt_5 and Ropt_3 yield similar NUN. Finally, Ropt_6 involves all features proposed in this work, it can minimize NUN, violations, wirelength and runtime. In particular, compared to the results for NTUplace in Table 4.5, the runtime for s2 is reduced from 48 hours to 13 hours; the number of violations in s19 is reduced from 15114 to 265.

By comparing Table 4.4 and Table 4.6, a big gap between global and detailed routing results can be found. Ropt_3 seems to get better results than NTUplace in Table 4.4, but it increases violations, wirelength and vias in its detailed routing results. In addition, the via improvement of Ropt_3 in Table 4.4 and Table 4.6 has a big mismatch because global routing model does not consider the vias generated by local routes. However, the vias generated by local routes are considerable. These imply that optimizing a placement for improving quality of its global routing result may not help in improving its effective routability in the detailed routing stage.

The top four placers in DAC12 placement contest, in alphabetical order, are mPL, NTUplace, Ripple, and SimPLR. To further evaluate the effectiveness of Ropt, we perform Ropt_6 to optimize the placement solutions of mPL, Ripple, and SimPLR in DAC12 contest. Tables 4.7, 4.8 and 4.9 reveal that Ropt_6 can reduce NUN, violations, total wirelength, via count and routing runtime in most placement solutions. Finally, Table 4.10 treats the detailed routing results of NTUplace as the baseline to compare the detailed routing results of mPL, NTUplace, Ripple, SimPLR and Ropt_6.

TABLE 4.7 COMPARING DETAILED ROUTING RESULTS OF MPL AND ROPT$_6$

	mPL					mPL+Ropt$_6$					
	NUN	Vio	WL$_d$(10^7)	Via$_d$(10^6)	R$_{cpu}$	NUN	Vio	WL$_d$(10^7)	Via$_d$(10^6)	R$_{cpu}$	P$_{cpu}$
s2	2068	113991	80.08	13.40	40:44:26	520	3033	76.79	12.65	18:00:13	00:25:15
s3	649	231	44.93	11.56	10:32:16	83	175	42.85	11.25	09:13:55	00:22:15
s6	892	231	45.42	12.22	10:13:03	125	243	43.92	12.00	09:36:14	00:20:49
s7	1525	170	55.55	18.44	14:13:09	153	144	53.13	17.93	12:28:56	00:29:16
s9	573	68	34.22	10.32	08:34:57	42	55	33.08	10.14	07:19:49	00:27:09
s11	531	11009	47.48	11.31	22:16:53	218	527	45.93	10.97	11:17:07	00:14:50
s12	994	343637	47.58	19.18	48:49:04	537	23309	44.69	18.71	20:16:41	00:39:59
s14	1717	271799	31.10	7.96	36:07:43	709	75677	30.17	7.72	15:47:54	00:13:43
s16	127	36	31.90	8.00	07:11:05	1	36	31.27	7.88	06:25:10	00:10:49
s19	811	276	20.74	6.43	05:57:55	164	358	19.89	6.27	05:51:20	00:25:31
Ratio$_{ind}$	**1**	**1**	**1**	**1**	**1**	**0.227**	**0.618**	**0.962**	**0.973**	**0.722**	
Ratio$_{sum}$	**1**	**1**	**1**	**1**	**1**	**0.258**	**0.140**	**0.961**	**0.972**	**0.568**	

TABLE 4.8 COMPARING DETAILED ROUTING RESULTS OF RIPPLE AND ROPT$_6$

	Ripple					Ripple +Ropt$_6$					
	NUN	Vio	WL$_d$(10^7)	Via$_d$(10^6)	R$_{cpu}$	NUN	Vio	WL$_d$(10^7)	Via$_d$(10^6)	R$_{cpu}$	P$_{cpu}$
s2	1743	81227	72.77	12.67	32:26:26	802	5928	70.18	12.26	23:01:26	00:25:32
s3	566	243	43.55	11.58	10:47:19	323	180	41.53	11.45	09:50:43	00:19:45
s6	267	232	40.96	11.77	10:35:16	146	235	40.07	11.83	09:34:46	00:18:26
s7	703	300	53.83	17.77	13:28:38	383	128	51.90	17.58	13:23:08	00:28:40
s9	125	8136	32.46	10.03	09:21:58	31	32	31.49	10.01	07:52:09	00:27:18
s11	115	433	40.22	10.48	11:14:55	89	428	38.70	10.42	10:50:42	00:13:37
s12	167	155	47.13	17.91	12:05:03	56	113	44.87	17.96	11:23:14	00:39:18
s14	1220	19086	27.79	7.59	11:16:24	961	10559	27.41	7.64	09:49:46	00:13:05
s16	129	38	29.17	7.92	06:13:12	41	50	29.11	8.04	06:16:49	00:10:51
s19	518	110	19.35	6.21	05:37:44	76	111	18.83	6.20	05:02:08	00:22:00
Ratio$_{ind}$	**1**	**1**	**1**	**1**	**1**	**0.473**	**0.685**	**0.970**	**0.996**	**0.904**	
Ratio$_{sum}$	**1**	**1**	**1**	**1**	**1**	**0.524**	**0.162**	**0.968**	**0.995**	**0.870**	

TABLE 4.9 COMPARING DETAILED ROUTING RESULTS OF SIMPLR AND ROPT$_6$

	SimPLR					SimPLR+Ropt$_6$					
	NUN	Vio	$WL_d(10^7)$	$Via_d(10^6)$	R_{cpu}	NUN	Vio	$WL_d(10^7)$	$Via_d(10^6)$	R_{cpu}	P_{cpu}
s2	553	876	69.67	12.47	14:37:14	170	686	67.65	11.99	13:37:42	00:23:23
s3	487	194	45.59	11.71	10:26:24	226	206	43.21	11.21	09:30:40	00:20:03
s6	443	361	41.54	11.80	10:26:33	71	236	40.56	11.63	09:26:07	00:17:41
s7	518	5402	55.44	18.15	15:40:00	302	170	50.91	17.46	12:08:27	00:29:15
s9	786	22	31.09	9.94	08:53:54	51	32	30.35	9.81	08:04:55	00:24:18
s11	979	1840	39.27	10.49	15:41:12	504	913	38.70	10.40	11:56:58	00:13:31
s12	715	241	46.98	17.80	12:04:46	277	117	44.89	17.76	11:06:29	00:37:47
s14	1459	224239	28.65	7.71	24:24:14	823	50725	28.05	7.53	12:51:33	00:12:23
s16	434	135	30.31	7.97	06:42:16	126	39	29.95	7.99	06:03:54	00:10:35
s19	510	72777	18.72	6.12	17:55:16	226	388	18.38	6.08	05:56:17	00:21:53
Ratio$_{ind}$	**1**	**1**	**1**	**1**	**1**	**0.378**	**0.549**	**0.968**	**0.982**	**0.787**	
Ratio$_{sum}$	**1**	**1**	**1**	**1**	**1**	**0.403**	**0.175**	**0.964**	**0.980**	**0.736**	

TABLE 4.10 COMPARISON BETWEEN THE DETAILED ROUTING RESULTS OF THE PLACEMENT SOLUTIONS IN DAC12 CONTEST.

	Ratio$_{ind}$					Ratio$_{sum}$				
	NUN	Vio	$WL_d(10^7)$	$Via_d(10^6)$	R_{cpu}	NUN	Vio	$WL_d(10^7)$	$Via_d(10^6)$	R_{cpu}
NTUplace	1	1	1	1	1	1	1	1	1	1
NTUplace+Ropt$_6$	0.322	0.483	0.987	1.002	0.759	0.325	0.016	0.985	1.002	0.634
Ripple	0.846	9.882	1.068	1.036	0.827	0.594	0.144	1.072	1.038	0.752
Ripple+Ropt$_6$	0.356	0.616	1.036	1.032	0.755	0.311	0.023	1.037	1.033	0.654
SimPLR	1.952	6.097	1.070	1.037	0.975	0.736	0.402	1.072	1.040	0.836
SimPLR+Ropt$_6$	0.751	0.891	1.035	1.018	0.748	0.297	0.070	1.033	1.019	0.615
mPL	2.082	369.029	1.153	1.078	1.378	1.058	0.973	1.155	1.083	1.249
mPL+Ropt$_6$	0.672	25.639	1.109	1.048	0.842	0.273	0.136	1.110	1.053	0.710

4.5.3 Comparison between Abacus and Our Legalizer

Table 4.11 shows the detailed comparison between Abacus used in Ropt$_1$ and our legalizer used in Ropt$_2$, in which D_{max}, D_{avg}, BD_{max} and BD_{avg} denote the maximum displacement, average displacement, maximum bin displacement and average bin displacement of cells after legalization, respectively. Notably, if the center of a cell before and after legalization is respectively located at G-cells b_i and b_j, the bin displacement of this cell is the index distance between b_i and b_j. Table 4.11 reveals that our legalizer obtains longer D_{avg} but shorter BD_{avg} than Abacus.

Traditional legalizers usually focus on minimizing D_{avg} to ensure consistency between the global placement solution and the legalized solution. Instead, our legalizer attempts to keep cells in the G-cells assigned by the global re-placement stage in order to preserve the global routing instance. Therefore, the average bin displacement is reduced. Tables 4.4 and 4.11 reveal that a legalizer minimizing average bin displacement may identify better placement solutions than minimizing average displacement in the routability-driven placement problem.

TABLE 4.11 COMPARISON BETWEEN ABACUS AND OUR LEGALIZER

	Abacus used in Ropt$_1$					Our legalizer used in Ropt$_2$				
	D_{max}	D_{avg}	BD_{max}	BD_{avg}	Time(sec)	D_{max}	D_{avg}	BD_{max}	BD_{avg}	Time(sec)
s2	306	11.025	10	0.742	2.98	333	17.738	11	0.138	4.08
s3	945	12.867	24	0.728	2.76	913	19.612	27	0.174	3.82
s6	840	13.830	24	0.749	2.82	836	20.649	24	0.168	4.14
s7	1391	13.944	43	0.721	3.6	1391	20.363	43	0.207	5.04
s9	691	12.680	22	0.754	2.21	691	19.604	22	0.190	2.87
s11	3089	13.594	83	0.771	2.09	3083	20.170	83	0.199	2.99
s12	595	12.285	17	0.791	4.5	595	19.935	17	0.159	7.53
s14	1292	14.770	38	0.718	1.43	1305	21.122	37	0.198	33
s16	872	13.421	22	0.738	1.86	872	20.210	22	0.139	2.86
s19	165	12.582	4	0.746	1.1	142	19.443	4	0.151	1.8
Ratio$_{ind}$	**1**	**1**	**1**	**1**	**1**	**1.088**	**1.609**	**1.100**	**0.185**	**1.369**
Ratio$_{sum}$	**1**	**1**	**1**	**1**	**1**	**0.998**	**1.518**	**1.010**	**0.231**	**2.688**

4.6 Summary

In this work, we first study the detailed routability of the placement solutions obtained by state-of-the-art routability-driven placers [57, 38, 39, 42]. Based on the observation of case study, we develop a routability optimizer Ropt that takes a placement solution and then optimizes its routability for both global routing and detailed routing. Ropt combines both placement and global routing. A global routing instance is built to provide the congestion information for placement algorithms. This work presents local-routability-aware global routing model, routing-cost-driven global re-placement, legalization with global routing preserved, and local detailed placement to optimize the routability. Finally, NCTU-GR 2.0 and Wroute are both adopted to evaluate the routability of Ropt's placement solutions. The experiment results obtained by both routers reveal that Ropt can improve routing congestion, wirelength and runtime of a given placement.

Chapter 5 NCTU-GR 2.0: Global Routing with Bounded-Length Maze Routing

5.1 Introduction

By holding global routing contests, ISPD07 and ISPD08 have attracted many researchers to develop robust and effective modern global routers. Table 5.1 shows the main features of each global router. Most of these routers apply the negotiation-based rip-up and rerouting, which is first introduced in PathFinder [22]. All of them focus on minimizing overflow first, and then wirelength and computation time. To lower computation time, most of modern global routers develop several efficient routing algorithms or acceleration approaches. Archer [10] proposed U-shaped pattern routing; NTUgr [12] presented escaping routing and FastRoute4.0 [16] developed 3-bend routing. Although many newly proposed routing algorithms can efficiently speed up path searching, maze routing is still the most important algorithm to seek a feasible or better connection after other routing algorithms fail in each global router even if it is relatively slow to explore the routing region as compared to other routing algorithms, such as pattern routing and monotonic routing algorithms. Most of modern global routers design various types of routing cost functions to balance total wirelength and total overflows. FGR [3] uses Lagrange multipliers and the negotiation technique to identify a routing cost function. NTHU-Route2.0 [8] dynamically adjusts the base cost to balance the used wirelength and remaining overflows. NCTU-GR [17] applies two-stage cost functions to accelerate early-stage routing and to enrich the capability of tackling hard-to-route nets. Although formally proposed routing cost functions avoid producing too much wirelength to eliminate overflows, they have no specific way to control the increase in wirelength. In this work, an optimal but slow bounded-length maze routing (BLMR) algorithm is proposed to identify the minimum-cost point-to-point path with wirelength below a length constraint. Besides, a heuristic BLMR algorithm is proposed to tackle the runtime issue of optimal BLMR algorithm.

TABLE 5.1 MAIN FEATURES OF MODERN GLOBAL ROUTERS

Global routers	Main features
FGR [3, 7]	Discrete-Largrange based routing cost, branch-free net topology
Maize-Router [8]	Extreme fast edge shifting
Box-Router 2.0 [9]	ILP-based box expansion
Archer [10]	U-shaped pattern routing
NTHU-Route 2.0 [11]	New history based cost function
NTUgr [12]	Multiple forbidden-region expansion, escaping routing
FastRoute [13-16]	Virtual capacity, 3-bend routing
NCTU-GR [17]	Pseudo Random Ordering, dual cost functions
GRIP [4, 5]	3D ILP-based global router performing on cluster-computing platform
MGR [6]	Multi-level 3D global routing

The RCE tool Grace presented in Chapter 2 can obtain a satisfactory global routing result in a short runtime budget. However, Grace cannot always continue to improve routing quality as the given runtime budget increases since Grace would spend too much routing resource at the beginning of routing stages such that Grace has no routing resource to refine the routing quality in the later routing stage. Accordingly, how to carefully use routing resource at the beginning of routing stages is a critical issue for global routers but not for RCE tools. In order to carefully use routing resource and thus yield better global routing results, we make the following contributions to global routing research in this work:

(a) This work addresses the BLMR problem, which involves identifying a minimal-cost path from a two-pin net's source to target with a specified length constraint. In the global routing problem, BLMR not only restricts the searching region by constraining every net's wirelength to speed up maze routing, but also avoids producing an overly long wirelength, conserving routing resources.

(b) To deal with a multi-terminal net routing, most global routers decompose each multi-pin net into several two-pin subnets, and then route each two-pin net individually, potentially diminishing the global view of optimizing a Steiner tree. The proposed RSMT-aware routing scheme can provide a global view to two-pin net routings, shortening the wirelength of multi-pin nets.

(c) The routing cost function is the common key for global routers to impact the routing quality. This work presents a dynamically adjusted history cost function to quickly learn which grid edges are critical. The routing resources on critical edges can be more carefully allocated to the nets that really demand these routing resources.

(d) The proposed NCTU-GR 2.0 invokes the abovementioned techniques and adopts different net ordering methods in different internal routing stages, **with a single set of control parameters for all benchmarks** to run faster and obtain shorter connections than the other state-of-the-art academic global routers.

The rest of this chapter is organized as follows. Section 2 reviews global routing problem. Section 3 presents BLMR algorithms, RSMT-aware routing scheme and dynamically adjusted history cost function. Section 4 presents NCTU-GR 2.0 based the proposed approaches. Section 5 summarizes the experimental results. Finally, Section 6 draws conclusions.

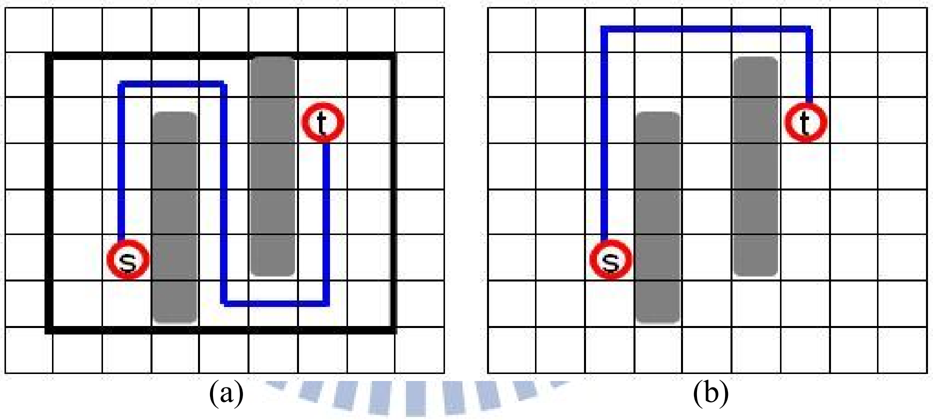


Fig. 5.1. (a) Maze routing within a bounding box; (b) maze routing without bounding box.

5.2 Problem Description

The global routing is formulated as the routing problem on a grid graph $G(V, E)$, where V denotes the set of G-cells, and E denotes the set of grid edges. Each grid edge is termed by the adjacency of the related G-cells to its two end nodes. The capacity $c(e)$ of a grid edge e indicates the number of routing tracks that cross the abutting boundary. The number of wires that pass through grid edge e is called the demand of the grid edge $d(e)$. The overflow of a grid edge e is defined as the amount of demand in excess of capacity. Moreover, the total overflow is the sum of overflows on all grid edges of E, and the maximum overflow is the maximum overflow among all edges. Given a set of nets, each of which is a set of pins. The objective of global routing problem is to find routing paths to connect each net, and minimize the maximum overflow, total overflow, total wirelength, and finally runtime.

5.3 Proposed Approaches to Improve Routing Quality

This section introduces three kernel techniques used in NCTU-GR 2.0, i.e., BLMR, RSMT-aware routing scheme, and dynamically adjusted history cost function.

5.3.1 BLMR

Maze routing is the most time-consuming process in modern global routers. Many global routers adopt a bounding box to limit the searching region of maze routing for acceleration. The initial bounding box is set to be slightly larger than the minimum rectangle that encloses all terminals of a net and, then, is gradually relaxed if the routing cannot be completed. However, maze routing within a bounding box may produce redundant wirelength and wastes routing resources. For instance, in Fig. 5.1, there are two overflow regions (dark regions). Maze routing within a bounding box is employed to connect source *s* to target *t* in Fig. 5.1(a), and maze routing without boundary constraint is employed in Fig. 5.1(b). The latter routing path is much shorter than the former. In this case, maze routing within a bounding box produces redundant wirelength and occupies extra routing resources, increasing the difficulty of identifying overflow-free routing paths in subsequent routings. Note that, because the proposed unilateral monotonic routing and HUM routing in Chapter 2 apply a bounding box for accelerating routing, their routing solutions may also have redundant wirelength. Accordingly, in NCTU-GR 2.0, we develop BLMR to speed up maze routing by limiting the routing length rather than the searching region while routing each two-pin subnet. Limiting the routing length not only can limit the searching region, but also can improve routing resource utilization by lessening redundant wirelength. The BLMR problem is formulated as follows. In a 4-tuple (s, t, G, L), s and t denote a source and a target, respectively; G represents the congestion graph; the grid edge in G has specified congestion cost; and L represents the bounded-length constraint (BLC) (L is not less than the Manhattan distance between s and t). The objective of the BLMR problem is to identify a minimal-cost path from s to t on graph G, and the wirelength of the path cannot exceed L. If the wirelength of the path is longer than L, a bounded-length violation is generated.

BLMR problem can be regarded as a restricted version of constrained shortest path (CSP) problem. In general CSP problem, a delay and a cost for every edge in a graph are specified. The delay of an edge may be the length of the edge or the signal latency from a terminal of the edge to another terminal. CSP algorithm attempts to identify a minimal-cost path from s to t, ensuring that the total delay of the identified path does not exceed an upper bound. The general version of CSP problem is NP-complete when the delay of each routing edge is a real number [24]. However, this problem can be solved in polynomial time if the delay of the each routing edge is an integer [25, 26]. Several studies [24-26] have addressed this problem for its applications in Quality of Service area.

Compared to CSP problem, BLMR problem owns some particular properties that are not in the general CSP problem. For instance, while CSP algorithm works on a general graph, BLMR algorithm works on a grid graph of specific 2D array structure with each vertex having at most four neighbors, which makes BLMR algorithm may be faster than CSP algorithm on solving global routing problem. Also, global routing problem owns the following particular properties:

1. The graph in global routing is a grid graph, and the distance from any node to the target can be estimated by simply Manhattan distance. Thus, a path possibly violating length constraint can be detected in the constant time.
2. The region to explore in a grid graph during routing can be restricted within a specified area; however, the studies [25, 26] scan entire routing graph, and thus consume much time.
3. In global routing, a net may be ripped up and re-routed several times to identify its final path. The paths of a net identified in two successive rerouting iterations make use of most routing edges in common. The proposed *history-based estimated wirelength* scheme utilizes this property.

Before detailing the proposed BLMR algorithms, the definitions of used notations are listed as follows. L denotes the bounded length, $P_i(s, v)$ denotes a path from source s to node v, $w_l(P_i)$ denotes the wirelength of P_i, $p_c(P_i)$ is the path cost of P_i, and $Manh(v, u)$ is the Manhattan distance from v to u. Notably, the proposed BLMR algorithm adopts A* search scheme for acceleration of path search. Therefore, $p_c(P_i)$ comprises the routed cost from s to the current node and the estimated lower bound cost from the current node to t.

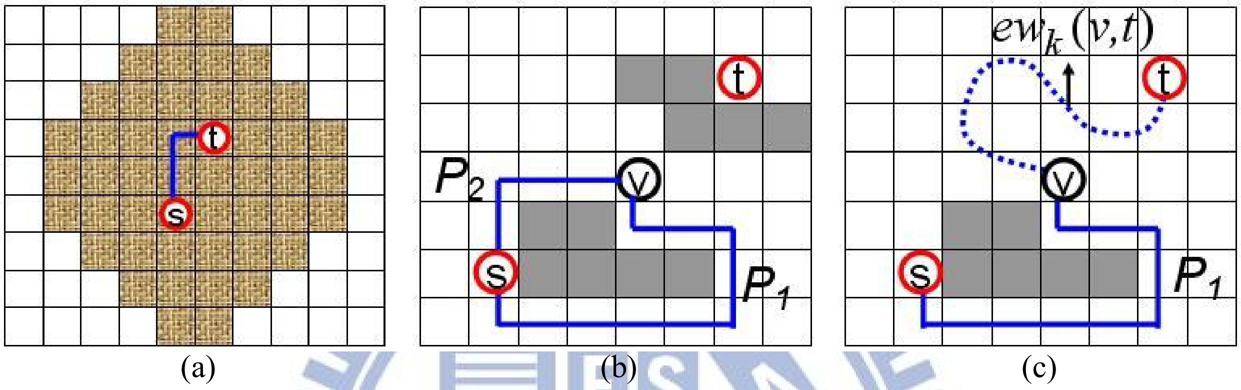


Fig. 5.2. (a) The search region of the net while L is set to 9; (b) two path candidates P_1 and P_2 from s to v; (c) $ew_k(v, t)$ represents estimating wirelength from v to t in iteration k

5.3.1.1 Optimal-BLMR

Optimal-BLMR algorithm adopts two different policies than traditional maze routing to obtain a minimum-cost routing solution under BLC. First, we define the potential wirelength (*pwl*) for each incomplete routing path $P(s,v)$ as the sum of $w_l(P(s,v))$ and $Manh(v,t)$ where v is an currently explored grid node, and a path with *pwl* exceeding L is regarded as a path violating BLC. Optimal-BLMR discards the paths violating BLC, and then restricts the searching region. Figure 5.2(a) shows the searching region of a net on the graph while L is set to 9. Second, assuming that there are two or more paths from s to v, traditional maze routing only preserves the minimum-cost path and discards others. However, in optimal-BLMR, this scheme does not guarantee to identify a feasible solution because the length slack of the minimum-cost path may be less than the wirelength required to detour around congested regions, where the length slack of $P(s,v)$ is L minus $w_l(P(s,v))$. For instance, Fig. 5.2(b) shows two path candidates P_1 and P_2 from s to v: the gray regions are congested regions, the bounded-length is 16, and $p_c(P_1)$, $p_c(P_2)$, $w_l(P_1)$ and $w_l(P_2)$ are 80, 90, 11 and 5, respectively. If optimal-BLMR only preserves the minimum-cost path P_1, the length slack of P_1 is 5, which is too small to detour around congested regions to reach *t*. Because the wirelength from *v* to *t* is uncertain before the end of routing, optimal-BLMR must preserve both paths. However, if the following inequalities hold, P_1 is considered to be inferior to P_2 and can be discarded.

$$w_l(P_1) \geq w_l(P_2) \quad \text{and} \quad p_c(P_1) \geq p_c(P_2)\,. \qquad (5.1)$$

S(*v*) is a *list* of node *v* to store the paths from the source *s* to node *v*, any two of which do not conform to Eq. (5.1). For each $v \in V$, *S*(*v*) is initially set as empty. While storing all currently explored paths, a Fibonacci heap *H* is initialized to have only *s*. At the beginning of each routing iteration, the minimum-cost path is selected from *H* for further routing. The optimal-BLMR algorithm is designed as follows.

1. Extract the minimum-cost path *P*(*s,v*) from *H*. If *v* is *t*, return *P*(*s,t*) as the solution and exit;
2. Explore each neighboring node of *v*, say node *u*. If the newly explored path *P*(*s,u*) does not conform to BLC, discard *P*(*s,u*); otherwise perform Step 3. Go back to Step 1 after exploring each neighbor of *v*.
3. Scan every path candidate in *S*(*u*) and remove the inferior paths to *P*(*s,u*) from *S*(*u*) and *H*. If *P*(*s,u*) is not inferior to any path in *S*(*u*), *P*(*s,u*) is inserted into *S*(*u*) and *H*.

Notably, if a net lacks adequate length slack to detour around all congestions to reach the target, optimal-BLMR would identify a routing path passing through congestions. A net passing through overflowed grid edges is called overflowed net, which will be rerouted in the next iteration.

The step 3 of optimal-BLMR takes the time complexity of $O(|S(u)|)$ to scan every path in *S*(*u*), where $|S(u)|$ denotes the number of path candidates in *S*(*u*). The maximum size of *S*(*u*) is derived as follows.

Lemma 1. *For an optimal-BLMR subject to the constraint* $Manh(s,u)+Manh(u,t) \leq L$*, where s, t, and u are respectively the source, the target and an intermediate node, the maximum size of* $S(u)$ *is* $\lceil (L - Manh(s,u) - Manh(u,t)+1)/2 \rceil$.

Proof. The shortest path candidate in $S(u)$ is the path of wirelength $Manh(s,u)$, and the longest path candidate in $S(u)$ must not exceed $L–Manh(u,t)$ due to BLC. Thus, the paths in $S(u)$ have at most $((L–Manh(u,t))–Manh(s,u)+1)$ possible wirelength levels. Moreover, because a detour increases the wirelength by two in unit of grid edge, the number of possible wirelength levels becomes $\lceil (L–Manh(u,t)–Manh(s,u)+1)/2 \rceil$. When Eq. (5.1) is adopted to prune the inferior paths, $S(u)$ only reserves the lowest cost path in each wirelength level. Hence, the maximum size of $S(u)$ is $\lceil (L–Manh(u,t)–Manh(s,u)+1)/2 \rceil$.

5.3.1.2 Heuristic-BLMR

Although the proposed heuristic-BLMR approach cannot guarantee the optimal solution, it is much faster than optimal-BLMR. The difference between optimal-BLMR and heuristic-BLMR is that heuristic-BLMR preserves only one path from the source to the current node, and the other paths are discarded. Accordingly, the major issue of heuristic-BLMR is to determine which path candidate should be preserved.

Path selection involves examining each path to determine whether or not the required wirelength to bypass the congested regions from the current node v to target t does not exceed the length slack, then heuristic-BLMR preserves the minimal-cost path with enough length slack. If no path candidates have enough length slack, the shortest path candidate is preserved because the shortest path has a greater chance to bypass the congested regions. However, the congestion information from v to t is not explored yet, hence the *history-based estimated wirelength* of the path from v to t is estimated as follows:

$$ew_k(v,t) = HL_{k-1}(s,t) \times \frac{Manh(v,t)}{Manh(s,t)} \tag{5.2}$$

where $ew_k(v,t)$ is the history-based estimated wirelength from v to t in iteration k (Fig. 5.2(c)), k denotes the iteration number of the NRR stage, and $HL_{k-1}(s,t)$ is the history length, i.e., actual routed wirelength from s to t in iteration k–1. The concept behind Eq. (5.2) is that the length from v to t is proportional to the length from s to t at previous iteration. Based on Eq. (5.2), heuristic-BLMR predicts that $P(s,v)$ has sufficient length slack to bypass the congested regions from v to t if the following equation holds,

$$w_l(P(s,v)) + ew_k(v,t) \le L. \qquad (5.3)$$

If multiple paths conform to Eq. (5.3), the minimum-cost path is preserved. If no path conforms to Eq. (5.3), the shortest path is preserved. This policy ensures that a path is only preserved to greatly reduce the number of explored routing paths during heuristic-BLMR. If Eq. (5.2) overestimates the wirelength from v to t in the previous iteration, the long path candidates tend to be discarded by Eq. (5.3), which selects the short path candidates as the final routing path. Hence the estimated wirelength from v to t using Eq. (5.2) at current iteration decreases. Similarly, if Eq. (5.2) underestimates the wirelength in the previous iteration, the estimated wirelength will increase in the current iteration. As a result, as the iteration number increases, Eq. (5.2) gradually becomes more accurate in estimating the actual wirelength from v to t.

This work evaluates the accuracy of Eq. (5.2) by the following experiment. As heuristic-BLMR identifies a routing path at iteration k, an internal node v is selected randomly in the identified path and, then, the difference between $HL_k(v,t)$ and $ew_k(v,t)$ is computed. According to the experiment on benchmark adaptec1, the average difference of all nets is 54% at the first iteration of the NRR stage; the average difference then gradually decreases to 30% at iteration five. Following iteration five, the average difference swings between 25% and 30%. However, if only the average difference of the first 20% long nets is calculated, the average difference ranges from 10% to 20% after iteration five, implying that Eq. (5.2) more accurately estimates long nets than short nets.

While closely resembling each other, the heuristic-BLMR algorithm and optimal-BLMR differ only in that the former always preserves at most one path in $S(v)$, $v \in V$. When $S(v)$ already contains a path and a new path $P(s,v)$ is explored, the path selection scheme proposed in this sub-section chooses the proper path to be preserved in $S(v)$ and H. Inaccurate estimation by Eq. (5.2) in heuristic-BLMR may yield unnecessary overflows, increasing the required iterations to remove unnecessary overflows.

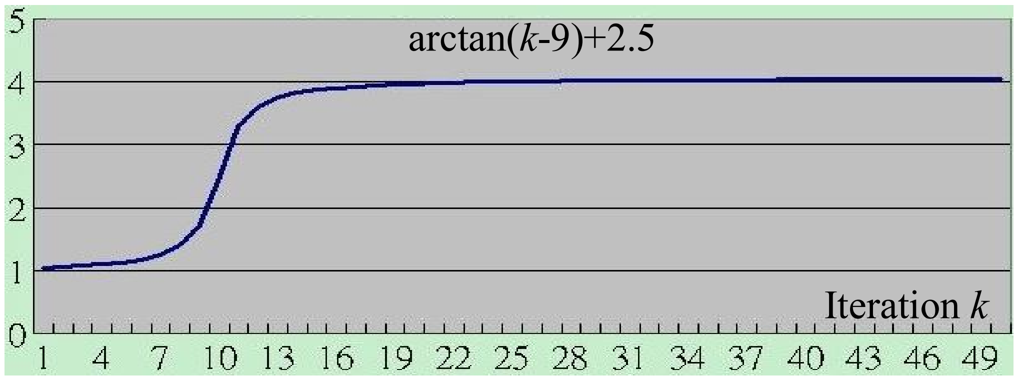


Fig. 5.3. Relationship between the routing iteration number and the scaling factor.

5.3.1.3 Bounded-Length Relaxation

In the NRR stage of this work, BLC is designed to control the routing resource utilization. Initially, the routed wirelength is strictly limited since overusing routing resources in the early stage likely inhibits subsequent routings from finding overflow-free paths, and increases runtime as well. While a routing cannot avoid congested regions under the strict BLC, BLC is gradually relaxed to encourage heuristic-BLMR to yield fewer overflows at the expense of a longer wirelength as the process iteration proceeds. However, overly relaxing BLC in a later stage only gives rise to a large increase in the wirelength, yet cannot help to resolve overflows. A bounded-length relaxation scheme is thus formulated as follows:

$$L_n^k = Manh(s_n, t_n) \times (1 + \arctan(k - \alpha) + \beta), \qquad (5.4)$$

where L_n^k is the bounded-length of two-pin net n in the k-th routing iteration. The first term is the Manhattan distance between two terminals of n, and the second term is the scaling factor of the bounded-length; α and β are user-defined positive constants. Figure 5.3 displays the relation between the routing iteration number and the scaling factor as α and β are set to 9 and 1.5, respectively. According to Fig. 5.3, the NRR stage can be roughly divided into three phases. The first phase is from iteration 1 to 8, in which the scaling factor increases roughly linearly. In phase 1, most overflows are removed and total wirelength increases slightly. The experiments on benchmarks [2] show that about 99.9% of total overflows are removed using only few seconds in this phase, and total wirelength only increases by 2.3%. In the second phase, the scaling factor increases rapidly from iteration 9 to 15. Although the searching region of each net is expanded by enlarged L_n^k, only a small number of nets must be rerouted. Accordingly, this phase also takes only little time. Most benchmarks of [2] are finished in this phase. In the final phase (iteration>16), the scaling factor grows very slowly to avoid producing an overly long wirelength and enlarging the search region too much.

The runtime and routing quality of the proposed router are impacted by the values of α and β. The value of α refers to the number of iterations in phase 1, while the value of β refers to the search region of each net. Increasing α encourages the proposed router using shorter wirelength to eliminate overflows, thus the proposed router can yield the final result with less wirelength at the cost of longer convergence period. Increasing β encourages an increase in the wirelength of the routing result with the number of routing iterations to be decreased. For the hard-to-route cases, large α and small β are effective for the proposed router to avoid producing too much wirelength and consuming additional routing resources at early stage, which increases the difficulty of eliminating overflows. For the easy-to-route cases, small α and large β are effective for the proposed router to complete the routings. In this work, α and β are set to 9 and 1.5, respectively.

5.3.2 RSMT-Aware Routing Scheme

Regarding the ability of the proposed scheme to decompose a net into multiple two-pin subnets, the proposed RSMT-aware routing scheme is characterized by RSMT and RMST. In the RMST decomposition stage, each net is first decomposed into several two-pin nets by RMST. Thereafter, the RSMT-aware routing scheme first constructs one RSMT for reference and then encourages the routing of each subnet to pass through the regions passed by RSMT. The terminals of a RSMT consist of pins and Steiner points. Two connected terminals are linked with a straight path or an L-shape path. For a RSMT of net N, a grid edge e is a *skeleton edge* associated with N if e is passed by a straight path of the RSMT. $SE(N)$ denotes the set of *skeleton edges* associated with N.

Figures 5.4(a)–5.4(d) illustrate the three steps to build RSMT-aware routing scheme for a four-pin net *N*. Three steps are performed at the stages of RMST decomposition and RSMT-aware scheme construction. Then, during the monotonic routing stage, the NRR stage and the post refinement stage, BLMR and monotonic routings use this scheme to evaluate the routing cost of grid edges. The details are listed as follows:

1. FLUTE and Kruskal's algorithm are first employed to yield a RSMT as the ideal routing tree and a RMST to decompose net *N* into several two-pin nets (Fig. 5.4(a)).
2. Identify skeleton edge set *SE*(*N*) (shadow regions in Fig. 5.4(b)) using the identified RSMT in the first step .
3. Each two-pin net of RMST is associated with *SE*(*N*) to form the RSMT-aware scheme. This association is used as follows. In Fig. 5.4(c). Net *N*'s RMST is split into three two-pin nets, n_1, n_2, and n_3. The edges in *SE*(*N*) will be assigned with less costs to encourage the routing wires of net *N* to pass through the edges in *SE*(*N*) (Fig. 5.4(d)) when the router performs monotonic or BLMR routing of n_1, n_2, and n_3.

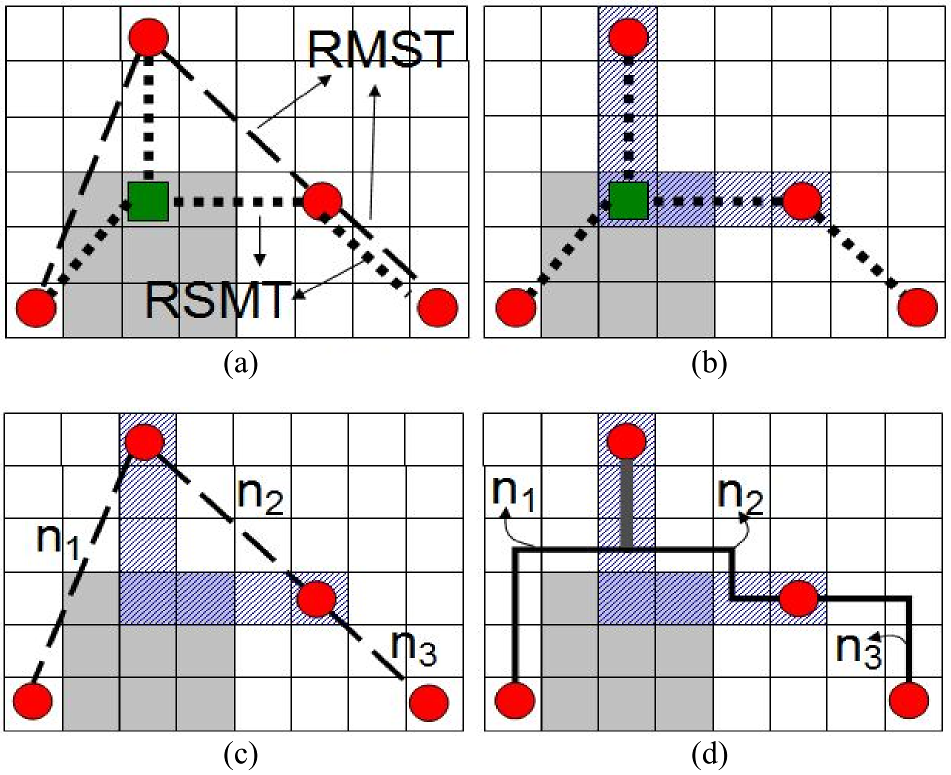


Fig. 5.4. (a) FLUTE and Kruskal algorithm are employed to yield one RSMT and one RMST; (b) the grid edges in the shadow regions are the skeleton edges of this net; (c) RMST is combined with *SE*; (d) the RSMT-aware routing result.

The proposed RSMT-aware routing scheme differs from other routers mainly in that, in the proposed scheme, a paragon RSMT is used for reference; the routing tree can then be easily amended to approach the paragon one as nearby regions become un-congested. Other routers normally refine the cost function to enable the subsequent routing to share the grid edges used in previous routing. However, for instance, if the path of n_1 in Fig. 5.4(d) is an upper L-shaped path, the next routing, e.g., n_2, cannot reuse the grid edges in the path of n_1. In addition, RSMT-aware routing scheme can restore the routing of a net to its RSMT at the post-refinement stage as congestions are eliminated. For instance, if the overflows (congested regions in gray color) in Fig. 5.4(d) are eliminated and subnets n_1 and n_2 are rerouted with RSMT-aware routing scheme, the routing path will pass through the Steiner point in Fig. 5.4(a) and look very similar to the RSMT in Fig. 5.4(a).

5.3.3 Dynamically Adjusted History Cost Function

With RSMT-aware cost function, a grid edge is assigned with different cost values to different nets to encourage a net to pass the grid edges in its skeleton edge set. For the routings of two nets, say N_1 and N_2, if grid edge e is in the skeleton edge set of N_1 but not in that of N_2, the proposed cost function assigns the cost of edge e for net N_1 as a value less than that for net N_2, which can better routing resource utilization by supporting each net to use the edges in its skeleton edge set. In the NRR stage, the concept of routing cost of grid edge e for net N is based on that in [11] and re-formulated as follows:

$$\text{cost}(e) = \begin{cases} 0 & \text{if } e \in GE(N) \\ (1+dah(e,k)) \times p_e + b_e - w & \text{elseif } e \in SE(N), \\ (1+dah(e,k)) \times p_e + b_e & \text{otherwise} \end{cases} \qquad (5.5)$$

where cost(e), b_e, and p_e are respectively routing cost of e, base cost of e and congestion penalty of e; $dah(e,k)$ denotes the dynamically adjusted history cost function (further discussed in Eq. (5.6)), $GE(N)$ denotes the set of grid edges that are passed by N, and w is a weighted constant that is set to 1 in this work. Equation (5.5) encourages a route to pass through the grid edges in either $GE(N)$ or $SE(N)$.

Traditionally, the history cost of a grid edge in negotiation-based cost function scheme (Eq. (1.2)) continues to increase if the grid edge keeps congested. The history cost remains unchanged even after the grid edge becomes un-congested, which results in overestimated routing cost for the routing passing the grid edge and then generates unnecessary detours. Hence, we have to lower the history cost if the grid edge becomes un-congested. The *dah*(*e,k*) is proposed as follows:

$$dah(e,k) = \frac{h_e^k}{C_1 + C_2 \times \sqrt{k}} \quad \text{where} \quad h_e^{k+1} = h_e^k + of_e^k, \qquad (5.6)$$

where h_e^k and of_e^k respectively denote the history cost and overflow frequency of *e* at iteration *k*. $h_e^1 = 1$, and h_e^k is updated at the end of every iteration. C_1 and C_2 are user-defined constants and set to 7 and 4, respectively. If edge *e* keeps un-congested during subsequent iterations, by Eq. (5.6), *dah*(*e*,*k*) will decrease as iteration number *k* increases while h_e^k remains unchanged. In addition, the proposed history cost is updated with overflow frequency instead of a constant value in traditional scheme (Eq. (1.2)). The overflow frequency of_e^k of grid edge *e* is set to zero at the beginning of each iteration *k*. Once an overflowed net is rerouted, of_e^k increases one if *e* overflows and *e* is passed by the rerouted net. A larger of_e^k implies a greater number of nets demanding the routing resource of *e* in iteration *k*, further implying that *e* is critical. The idea behind Eq. (5.6) is that critical routing resources should have large overflow frequency. Thus, a grid edge with high overflow frequency is assigned with large history cost. As compared to traditional history cost updating scheme, the proposed one requires less iterations to distinguish which grid edges are critical. In this work, p_e and b_e are formulated as follows, which are inspired by [13] and [11], respectively.

$$p_e = 1 + \frac{C_3}{1 + \exp(C_4(c(e) - d(e)))} \text{ and } b_e = C_5 + C_6 / 2^k, \qquad (5.7)$$

where C_3, C_4, C_5 and C_6 are set to 150, 0.3, 30 and 200 in our implementation, respectively. The constant value setting is determined by closely examining how their variations impact routing results in the experiments.

5.4 Design Flow of NCTU-GR 2.0

Figure 5.5 shows the design flow of NCTU-GR 2.0. At first, the 3D routing problem is compacted into a 2D routing problem. Then, each net is decomposed to two-pin nets based on the topology of

RMST, and skeleton edge sets are built for each multi-pin net to set up RSMT-aware routing scheme. Moreover, an initial congestion graph is generated via monotonic routing that routes all two-pin nets.

In each iteration of the NRR stage, all two-pin nets are first sorted in an array, and then the two-pin nets are examined sequentially to determine whether pass through overflowed grid edges. For the overflowed net e.g. *N*, BLC of *N* is relaxed by Eq. (5.4) ,and then is ripped up and rerouted by heuristic-BLMR with the routing cost function in Eq. (5.5), which requires testing if a grid edge belongs to *GE*(*N*) or *SE*(*N*). *GE*(*N*) and *SE*(*N*) are stored using hash table, specifically, *unordered_hash* in C++ STL. *SE*(*N*) is constructed in the RMST decomposition stage while *GE*(*N*) is dynamically updated during rip-up and rerouting to record the grid edges in the routing path of net *N*. As NRR enters a new iteration, the history cost is updated by Eq. (5.6). the NRR stage iteratively reroutes the overflowed nets until an overflow-free routing result is obtained.

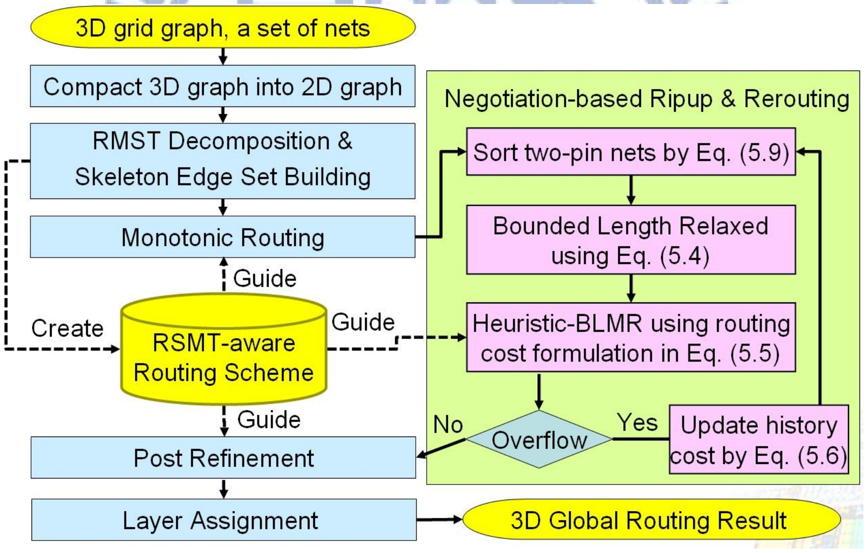

3D grid graph, a set of nets
Compact 3D graph into 2D graph
RMST Decomposition & Skeleton Edge Set Building
Monotonic Routing
Guide
Create
RSMT-aware Routing Scheme
Guide
Guide
Post Refinement
Layer Assignment
Negotiation-based Ripup & Rerouting
Sort two-pin nets by Eq. (5.9)
Bounded Length Relaxed using Eq. (5.4)
Heuristic-BLMR using routing cost formulation in Eq. (5.5)
No
Overflow
Yes
Update history cost by Eq. (5.6)
3D Global Routing Result

Fig. 5.5. Design flow of NCTU-GR 2.0.

The wirelength of each net is then greedily minimized in the post refinement stage by ripping up and rerouting each two-pin net once with the proposed RSMT-aware routing scheme. The overflow-free two-pin routings without a detour are rerouted by monotonic routing; meanwhile, the other two-pin routings are rerouted by heuristic-BLMR with the original path length as BLC. Rerouting overflow-free nets can reduce their wirelength and can vacate the routing resource to other overflowed nets as well. Because most nets are routed by monotonic routing or heuristic-BLMR with small BLC, this stage is very efficient. In this stage, the routing cost of a grid edge, say e, is formulated as follows:

$$\text{cost}(e) = 1 + \kappa(\frac{d(e)}{1 + c(e)}) . \tag{5.8}$$

If $c(e) \le d(e)$, κ is set to 10^5 to avoid increasing overflows; otherwise, κ is set to 0.1. Finally, the layer assignment in [17] is employed to transform the 2D routing result to the 3D result.

TABLE 5.2 . NET ORDERING METHODS COMPARISION

Order	adaptec1		adaptec3		adaptec5		Average	
	WL (10^5)	CPU (min)	WL (10^5)	CPU (min)	WL (10^5)	CPU (min)	WL (10^5)	CPU (min)
LenD	36.42	2.17	96.00	1.93	105.18	4.65	70.97	2.18
LenI	36.65	2.26	96.28	1.85	106.73	5.37	71.53	2.31
OFD	36.39	2.13	96.02	1.86	105.39	4.46	70.99	2.10
OFI	36.74	2.51	96.42	1.98	107.02	5.63	71.69	2.46
Eq. (5.9)	36.31	1.78	95.97	1.77	105.14	4.06	70.88	1.90

The routing ordering of two-pin nets impacts the routing quality and runtime. We introduce the net routing ordering adopted in each routing stage as follows. In the monotonic routing stage, the proposed router sorts the two-pin nets in the increasing order according to its bounding box size. Smaller bounding box for a net implies less solution space for monotonic routing of the net. Routing a net with less solution space earlier can improve the possibility to complete its monotonic routing due to fewer previously routed wires. In the NRR stage, wirelength and congestion control and overflow reduction are main objectives. We evaluate four net ordering methods in terms of wirelength and overflow in the NRR stage. Table 5.2 displays the global routing wirelength (WL) and runtime (CPU) of benchmarks adaptec1, adaptec3, adaptec5 and average of all overflow-free cases when the NRR stage adopts four different net ordering in decreasing wirelength (LenD), increasing wirelength (LenI), decreasing overflows (OFD) and increasing overflows (OFI). Notably, WL here does not include vias because layer assignment has not yet been performed. Table 5.2 reveals that routing the long two-pin nets with more overflows earlier can identify the results with shorter wirelength and runtime. Based on this observation, all two-pin nets are sorted at the beginning of each iteration in the NRR stage, based on nets' score (Eq. (5.9)) in decreasing order.

$$score_k(n) = C_7 \times oe_{k-1}(n) + C_8 \times len_{k-1}(n), \qquad (5.9)$$

where $score_k(n)$ denotes the score of two-pin net n in iteration k, $oe_{k-1}(n)$ and $len_{k-1}(n)$ denote the number of overflowed grid edges passed by n and the length of n in iteration k–1, respectively. Notably, $oe_0(n)$ and $len_0(n)$ depend on the routing outcome of the monotonic routing stage. Additionally, C_7 and C_8 are user defined constants and set as 30 and 1 in this work. The bottom row in Table 5.2 displays the routing results as the NRR stage adopts Eq. (5.9) to sort two-pin nets, which obtains a shorter wirelength and runtime than other ordering methods. Finally, the post refinement stage adopt wirelength-decreasing net ordering since most overflows have been removed.

5.5 Experimental Results

The proposed algorithms were implemented in C/C++ language on an 8-core 3.0 GHz Intel Xeon-based server with 32GB memory. ISPD08 global routing benchmark circuits [2] were used in our experiments. We classify the benchmarks into two types, i.e., overflow-free cases and hard-to-route cases. All state-of-the-art global routers cannot identify an overflow-free routing result for each hard-to-route case. As for overflow-free cases, most state-of-the-art routers can identify an overflow-free routing result for each one. To overflow-free cases, the routers in the following experiments perform until an overflow-free outcome is achieved; to hard-to-route cases, the routers stop when overflows are not improved in five successive iterations.

5.5.1 Comparing Traditional Maze Routings with BLMR

To compare traditional maze routings with and without bounding box and BLMR, we implement three global routers with different maze routing approaches in the NRR stage. MR-GR, MRB-GR and H-BLMR-GR denote three different global routers, where MR-GR employs maze routing without using bounding box; MRB-GR employs bounding box to limit the search region, and H-BLMR-GR adopts the proposed heuristic-BLMR and bounded-length relaxation scheme in the NRR stage. The initial bounding box used by MRB-GR extends by 10 units of grid edges four boundaries of the minimum rectangle enclosing all terminals of the routed net. If an overflow-free path cannot be obtained within the bounding box, each boundary of the bounding box is extended by 10 units of grid edges again in the next iteration. This bounding box expansion scheme is also used in [14]. Notably these routers adopt the same routing cost function and the proposed RSMT-aware routing scheme is not used by these routers. Table 5.3 shows the routing results of these routers, where WL and CPU are total wirelength and CPU time, respectively. Table 5.3 indicates that MRB-GR is faster than MR-GR, but MRB-GR produces more wirelength than MR-GR because MRB-GR may detour often to avoid congested regions. H-BLMR-GR produces less wirelength than MR-GR and MRB-GR because it has less detours than MR-GR and MRB-GR. The BLC of H-BLMR-GR restricts the searching region such that H-BLMR-GR is faster than MR-GR and MRB-GR. Furthermore, the proposed bounded-length relaxation scheme offers more efficient routing resource utilization than the other two routers. As a result, H-BLMR-GR can eliminate all overflows of newblue1 but MR-GR and MRB-GR cannot.

TABLE 5.3 ROUTING RESULT COMPARISON BETWEEN MAZE ROUTING W/ AND W/O BOUNDING BOX AND BOUNDED-LENGTH MAZE ROUTING

ISPD'08 benchmark	MR-GR		MRBB-GR		H-BLMR-GR	
	WL	CPU(min)	WL	CPU(min)	WL	CPU(min)
adaptec1	53.86	5.90	54.29	3.85	53.55	2.65
adaptec2	52.14	3.36	52.49	0.96	51.97	0.79
adaptec3	131.16	4.47	131.85	3.72	129.88	3.50
adaptec4	120.88	1.32	120.98	1.25	120.76	1.20
adaptec5	155.76	14.66	157.63	9.20	155.52	9.16
newblue1	x	x	x	x	46.26	2.80
newblue2	74.74	0.70	74.77	0.67	74.63	0.63
newblue5	231.33	53.94	233.45	18.73	230.45	7.22
newblue6	178.71	17.66	179.69	6.23	177.23	5.46
bigblue1	57.17	10.69	59.02	6.13	57.61	8.49
bigblue2	90.05	48.40	89.90	29.01	89.42	7.73
bigblue3	129.95	15.79	130.15	2.44	129.68	2.19
Ratio	1	1	1.007	0.575	0.997	0.494

5.5.2 The Effectiveness of RSMT-aware Routing

Table 5.4 shows the effectiveness of RSMT-aware routing scheme, in which the second, third and fourth (fifth, sixth and seventh) columns show the wirelength, routing iterations and runtime of H-BLMR-GR without (with) RSMT-aware routing scheme, respectively. H-BLMR-GR with RSMT-aware routing scheme reduces 0.825% wirelength than that without the scheme. Although RSMT-aware routing scheme spends additional effort to identify RSMT, less wirelength usage makes H-BLMR-GR demand less iterations and then converge faster. The iteration number and runtime of H-BLMR-GR with RSMT-aware routing scheme are reduced by 18.32% and 19.58%, respectively, than that without the scheme.

TABLE 5.4 COMPARISON OF THE ROUTING RESULT OF H-BLMR-GR WITH AND WITHOUT RSMT-AWARE ROUTING SCHEME

ISPD'08 benchmark	H-BLMR-GR(w/o RSMT)			H-BLMR-GR(w/ RSMT)		
	WL	Rounds	CPU (m)	WL	Rounds	CPU (m)
adaptec1	53.55	11	2.65	53.04	10	2.52
adaptec2	51.97	12	0.79	51.51	10	0.65
adaptec3	129.88	9	3.50	129.24	9	3.40
adaptec4	120.76	7	1.20	120.53	8	1.23
adaptec5	155.52	16	9.16	154.01	11	6.03
newblue1	46.26	83	2.80	45.76	67	2.37
newblue2	74.63	5	0.63	74.51	5	0.64
newblue5	230.45	21	7.22	228.68	15	5.26
newblue6	177.23	14	5.46	175.49	11	4.44
bigblue1	57.61	18	8.49	56.29	9	4.09
bigblue2	89.42	86	7.73	88.66	48	4.21
bigblue3	129.68	22	2.19	129.35	19	1.91
Improve				0.825%	18.3%	19.58%

5.5.3 Comparison of Optimal-BLMR and Heuristic-BLMR

Table 5.5 compares the routing results of optimal-BLMR, heuristic-BLMR and CSP algorithm in [25]. The BLMR problem is a restricted version of CSP, so the algorithm of [25] can be adopted to solve BLMR problem. The algorithm of [26] can identify the optimal solution of BLMR problem via dynamic-programming technique. The time complexity of the CSP algorithm in [25] is O(|E||L|) where |E| is the number of grid edges in the routing graph and L is BLC. O-BLMR-GR and CSP-GR adopts the optimal-BLMR and the CSP algorithm [25] in the NRR stage, respectively. Note that the routers in Table 5.5 all employ RSMT-aware routing scheme. Table 5.5 shows that H-BLMR-GR runs averagely 269.21 times faster than O-BLMR-GR, and only increases 0.1% total wirelength. The experiments indicate that heuristic-BLMR can take much less runtime to yield similar routing results with optimal-BLMR. On the other hand, the routing results of CSP-GR and O-BLMR-GR are same but the runtime of CSP-GR is much larger than that of O-BLMR-GR since CSP algorithm [25] does not take three properties of global routing into account, as described in section 3.

TABLE 5.5 COMPARISION OF THE ROUTING RESULT OF GLOBAL ROUTERS WITH HEURISTIC-BLMR, OPTIMAL-BLMR AND [27]

ISPD'08 benchmark	H-BLMR-GR (w/ RSMT)		O-BLMR-GR (w/ RSMT)		CSP-GR (w/ RSMT)	
	WL	CPU (m)	WL	CPU (m)	WL	CPU (m)
adaptec1	53.04	2.52	52.93	615.21	52.93	3586.67
adaptec2	51.51	0.65	51.49	18.75	51.49	61.31
adaptec3	129.24	3.4	129.09	777.14	129.09	7406.14
adaptec4	120.53	1.23	120.52	30.48	120.52	159.11
adaptec5	154.01	6.03	153.66	1607.43	153.66	10126.81
newblue1	45.76	2.37	45.68	1044.53	45.68	7551.95
newblue2	74.51	0.64	74.50	5.68	74.5	33.63
newblue5	228.68	5.26	228.55	1766.12	228.55	12009.62
newblue6	175.49	4.44	175.44	664.18	175.44	5074.34
bigblue1	56.29	4.09	56.04	2537.78	56.04	11952.94
bigblue2	88.66	4.21	88.56	2576.10	88.56	13447.24
bigblue3	129.35	1.91	129.29	516.24	129.29	4315.77
ratio	1	1	0.999	269.21	0.999	1705.67

5.5.4 Routing Result Comparison of Sequential Routers

H-BLMR-GR with RSMT-aware routing scheme includes all innovation presented in this work, so we re-name H-BLMR-GR with RSMT-aware routing scheme to NCTU-GR 2.0. Tables 5.6 and 5.7 compare the routing results of NCTU-GR 2.0 with four state-of-the-art global routers. NTHU-Route2.0

[11], FastRoute 4.1 [16] and NCTU-GR [17] are 2D router with layer assignment. MGR is a multi-level 3D router which runs much faster than traditional 3D routers [3-5]. The results of MGR are quoted from [6] because the binary of MGR is unavailable. MGR performs on a 2.6GHz Intel CPU with 16G memory while the other routers are all performed on our platform, so the runtime of MGR is normalized by the clock rate ratio 1.154. Notably, various control parameters in routers affect the routing quality and performance. **NCTU-GR 2.0 and FastRoute4.1 adopt a single set of control parameters to solve all benchmarks while NCTU-GR, NTHU-Route2.0 use different control parameters to identify their best routing result for each benchmark.** MGR automatically adapts parameters based on the characteristics of each benchmark. For comparison, we also adopt different sets of control parameters to yield the best routing result for each benchmark, listed at column NCTU-GR$_B$ 2.0 of Tables 5.6 and 5.7.

Table 5.6 compares each router by overflow-free cases, in which the wirelength and runtime are the primary items for comparison because all routers produce overflow-free routing results. In Table 5.6, NCTU-GR 2.0 identifies 1.1%, 1.1% and 0.6% less wirelength and averagely runs 1.90×, 1.77× and 1.92× faster than NTHU-Route2.0, FastRoute4.1 and NCTU-GR, respectively. Compared to MGR, NCTU-GR 2.0 identify 0.3% longer wirelength; but NCTU-GR$_B$ 2.0 can achieve shorter wirelength than MGR.

For hard-to-route cases, the routers in [6, 11, 16] focus on minimizing total overflows, since total overflow provides a more global perspective on congestion information than the maximum overflow. However, the inability to address the maximum overflow may lead to very congested hot spots, possibly becoming unroutable for detailed routing. Therefore, this work considers both total overflow and maximum overflow. NCTU-GR 2.0 regards maximum overflow (MO) as the first minimization objective, and total overflow (TO) as the second objective. Table 5.7 reveals that NCTU-GR 2.0 achieves good performance for maximum overflow, wirelength and runtime, but the total overflow still has room for improvement. As for the best routing result for each benchmark (NCTU-GR$_B$ 2.0), the proposed router performs very well in the total overflow and runtime. Notably, the MO information of MGR is unavailable, so we do not list MO of MGR in Table 5.7.

TABLE 5.6 COMPARISON BETWEEN NCTU-GR 2.0 AND THE OTHER ROUTERS ON OVERFLOW-FREE CASES.

Benchmark	NCTU-GR 2.0		NCTU-GR$_B$ 2.0		NTHU-Route2.0		FastRoute4.1		NCTU-GR		MGR	
	WL	CPU(m)	WL	CPU(m)	WL	CPU(m)	WL	CPU(m)	WL	CPU(m)	WL	CPU(m)
adaptec1	53.04	2.52	52.35	2.30	53.49	4.86	53.73	3.31	53.50	3.90	52.82	4.39
adaptec2	51.51	0.65	51.30	0.64	52.31	1.42	52.17	0.95	51.69	1.45	51.46	1.04
adaptec3	129.24	3.40	128.34	2.96	131.11	6.16	130.82	3.69	130.35	4.88	128.92	4.83
adaptec4	120.53	1.23	120.17	1.18	121.73	2.08	121.24	1.25	120.67	2.28	119.96	1.41
adaptec5	154.01	6.03	151.85	4.97	155.55	11.95	155.81	6.70	154.70	9.07	153.23	7.95
newblue1	45.76	2.37	45.62	1.93	46.53	4.07	46.33	12.01	45.99	3.63	45.58	4.51
newblue2	74.51	0.64	74.51	0.63	75.85	1.17	75.12	0.85	74.88	0.90	74.46	0.80
newblue5	228.68	5.26	225.94	4.62	231.73	10.88	230.94	9.82	230.31	15.03	228.00	6.54
newblue6	175.49	4.44	171.10	4.02	177.01	10.34	177.87	8.78	176.87	9.67	174.86	7.04
bigblue1	56.29	4.09	55.33	3.44	56.35	6.93	56.64	4.22	56.56	6.35	55.82	5.04
bigblue2	88.66	4.21	86.71	3.45	90.59	6.47	91.18	12.12	89.40	11.18	88.92	6.00
bigblue3	129.35	1.91	127.67	1.78	130.76	3.91	130.04	2.06	129.66	4.38	128.75	2.89
Ratio	1	1	0.989	0.894	1.012	1.900	1.012	1.768	1.006	1.920	0.997	1.45

TABLE 5.7 COMPARISON BETWEEN NCTU-GR 2.0 AND THE OTHER ROUTERS ON HARD-TO-ROUTE BENCHMARKS.

	NCTU-GR 2.0				NCTU-GR$_B$ 2.0				NTHU-Route 2.0			
	MO	TO	WL	CPU(m)	MO	TO	WL	CPU(m)	MO	TO	WL	CPU(m)
newblue3	194	31710	105.36	143.34	194	31526	106.80	63.34	204	31454	106.49	64.97
newblue4	2	144	127.27	17.33	2	132	129.27	17.48	4	138	130.46	52.01
newblue7	2	58	342.9	85.67	2	54	341.90	74.53	2	62	353.35	50.28
bigblue4	2	194	225	60.23	2	132	227.10	63.55	2	162	231.04	52.63
ratio	1	1	1	1	1.00	0.88	1.01	0.84	1.263	0.964	1.023	1.229

	FastRoute 4.1				NCTU-GR				MGR			
	MO	TO	WL	CPU(m)	MO	TO	WL	CPU(m)	MO	TO	WL	CPU(m)
newblue3	736	31276	108.4	15.99	198	31808	104.28	131.43	X	31026	107.22	19.99
newblue4	2	136	130.46	65.23	2	134	126.79	40.92	X	136	128.54	15.64
newblue7	4	54	353.38	868.74	2	114	338.63	71.52	X	56	349.02	110.12
bigblue4	2	130	230.24	93.25	2	164	223.99	65.37	X	134	225.73	21.31
ratio	1.948	0.882	1.027	3.891	1.005	1.186	0.992	1.3	X	0.90	1.01	0.67

5.6 Summary

This chapter at first discusses the algorithmic differences between designing a global-routing-based RCE and a global router, and then points out that a global router should carefully use routing resource for obtaining high-quality routing results. Accordingly, this work proposes bounded-length maze routing algorithms, RSMT-aware routing scheme, and dynamically adjusted history cost function to better the usage of routing resource. Moreover, a global router NCTU-GR 2.0, characterized by abovementioned approaches, is presented. Excellent performance of NCTU-GR 2.0 on ISPD benchmarks with a single set of parameters is owing to systematic and effective wirelength control, flexible tree structure change, and valid update of edge congestion status with overflow frequency.

Chapter 6 Post3DGR: Post Optimization of 3D Global Routing Results

6.1 Introduction

In advanced technology nodes, designs contain considerable metal layers. To our knowledge, a design in 28*nm* technology node may have up to 15 layers. As the number of metal layers increases, the limitations of traditional global routing flows emerge. Generally, two routing flows are adopted to resolve the global routing problem. The first one directly performs global routing on a 3D grid graph to generate a 3D routing result [3-6]. The other condenses the 3D grid graph into the 2D grid graph and, then, adopts a 2D global router to obtain a 2D routing result; the layer assignment finally assigns each net edge in the 2D routing result to a layer in the 3D grid graph. The limitation of the first flow is too slow, while the limitation of the second flow is the lack of layer and via information in its 2D routing stage.

Most global routers [7-21] adopt the flow of 2D routing with layer assignment due to the runtime concern. However, the lack of layer and via information in the 2D routing stage would jeopardize the results' quality. For example, Figs. 6.1(a) and 6.1(b) show two path candidates of a net of two pins highlighted by red vertices, while Figs. 6.1(c) and 6.1(d) depict the layer assignment results of Figs. 6.1(a) and 6.1(b), respectively, in which the preferred routing direction of layers 2 and 4 is horizontal while others are vertical. Since grid edges $e_{a,2}$, $e_{b,4}$, and $e_{c,4}$ have no routing resource due to blockages or congestion, the routing path in Fig. 6.1(c) needs to a z-axis detour to bypass $e_{a,2}$, $e_{b,4}$, and $e_{c,4}$. In this example, the better path in the 2D grid graph (Fig. 6.1(a)) appears to yield a worse path in the 3D grid graph (Fig. 6.1(c)). This example shows that directly exploring the 3D grid graph has potential to identify a better global routing result than 2D grid routing with layer assignment, and thus warrants the algorithm development that optimizes 3D global routing results on the 3D grid graph.

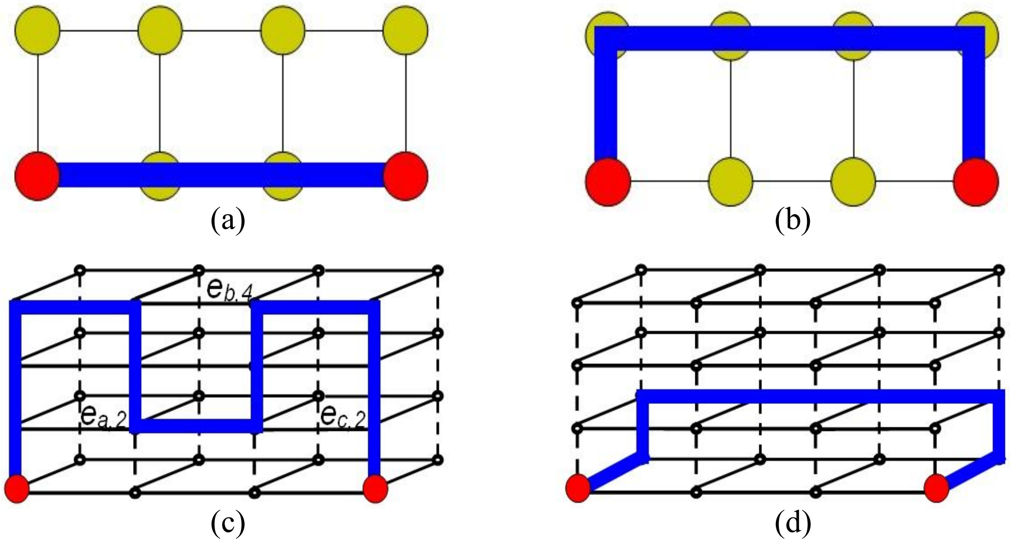


Fig. 6.1. Gap of the recognition of good results between 2D routing with layer assignment and 3D routing. (a) a good 2D routing result; (b) a bad 2D routing result; (c) a bad 3D routing result; (d) a good 3D routing result.

GRIP [4] is an ILP-based 3D parallel global router performing on a cluster computing environment. To our knowledge, despite obtaining the routing results of the best wirelength in the literature, GRIP consumes a considerably higher runtime than other global routers, GRIP [4] is about 270X slower than NCTU-GR 2.0. Although the work in [5] better exploits the computation power of cluster computing environment to speed up GRIP, the runtime of GRIP is still 42X slower than NCTU-GR 2.0. To reduce the runtime of routing on 3D grid graph, a 3D global router MGR is based on a multi-level framework. However, the coarsening stage in the multi-level framework more or less hides some information in the original grid graph, possibly degrading the solution quality. For example, a net of two pins (red vertices) is routed on a 4x4 grid graph with three obstacles (gray rectangles) in Fig. 6.2(a). In Fig. 6.2(b), the coarsening stage of MGR merges four neighboring grid nodes into a super node to shrink the 4x4 graph to a 2x2 graph; a routing path passing thorough node v_1 is then identified because the congestion in v_1 is lower than that in v_2. The un-coarsening stage maps the routing path shown in Fig. 6.2(b) back to the original 4x4 grid graph to obtain the routing result shown in Fig. 6.2(c), which is a suboptimal result as compared to the shortest path in Fig. 6.2(d), due to some information hidden at the coarsest level.

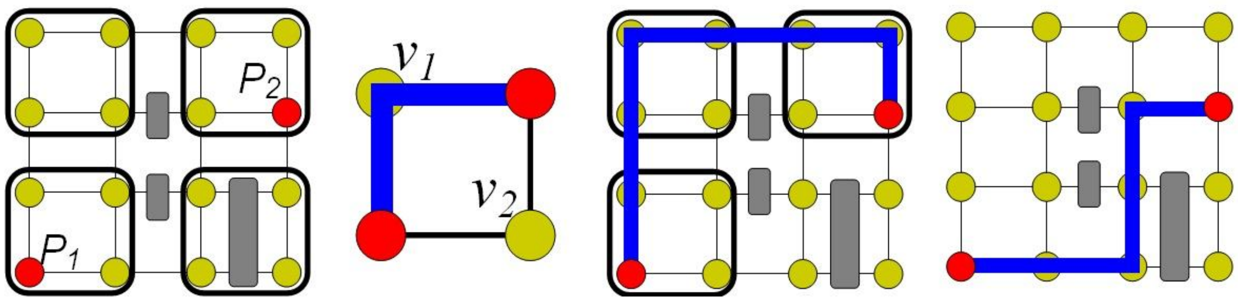


Fig. 6.2. MGR flow. (a) a two-pin net routes on a 4x4 graph; (b) the coarsening stage routes the net on a coarsened 2x2 graph; (c) the un-coarsening stage maps the routing path in (b) back to the original graph; (d) the shortest path of (a).

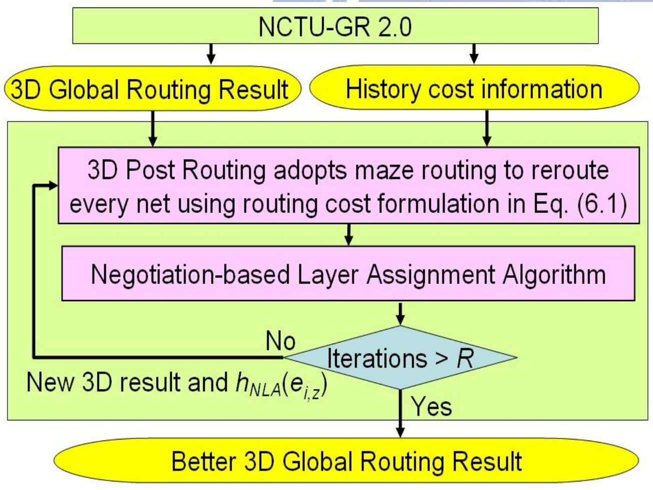
Current research addresses the global routing problem from three aspects, 2D global routing, 3D global routing, and layer assignment. Most relevant research examines these aspects individually without discussing the necessity to coordinate 2D routing, 3D routing, and layer assignment in a unified flow. This work presents a unified global routing framework to efficiently generate a 3D global routing result by 2D global routing with layer assignment, and then adopts the proposed post-3D-global-routing tool **Post3DGR** to reduce the congestion, wirelength and via count of the 3D global routing result. Post3DGR combines 3D routing and layer assignment to iteratively refine the 3D global routing result. The proposed unified global routing framework can diminish the large gap in runtime and quality between 2D and 3D routings. This framework can yield a slightly better quality than that of GRIP of the best quality so far, but consumes markedly less runtime than GRIP.

In addition to reduce vias, wirelength and congestion, Post3DGR can be extended to address some manufacturing issues that are hard considered in 2D global routing stage due the lack of layer and via information. Well addressing the manufacturing issues before detailed routing simplifies the subsequent detailed routing. The work in [30] takes the double patterning issue into account; the works in [32, 33] consider antenna effect to improve yield, and the works in [28, 29] focus on minimizing via overflows. In section 5 of this chapter, we will illustrate the efficiency of extending Post3DGR to consider antenna effect.

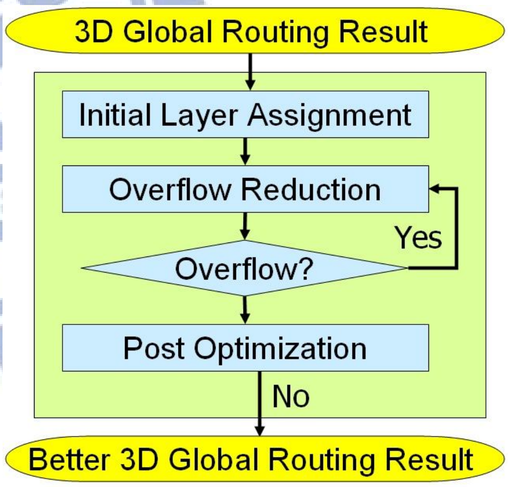
The rest of this chapter is organized as follows. Section 2 formulates the problem of optimizing a 3D global routing result. Section 3 describes the framework of Post3DGR. Section 4 presents the proposed negotiation-based layer assignment that is the kernel algorithm in Post3DGR. Section 5 summarizes the experimental results. Conclusions are finally drawn in Section 6.

6.2 Problem Description

Given a 3D global routing result which consists of a k-layer 3D grid graph $G^k(V^k, E^k)$ and a set of routing solutions of nets, the objective of Post3DGR is to refine the 3D global routing result to further minimize overflows, wirelength and vias. In 3D grid graph $G^k(V^k, E^k)$, V^k denotes the set of 3D grid nodes; each grid node refers to a G-cell; and E^k refers to the set of 3D grid edges, each of which is termed by the adjacency of the related G-cells of its two end nodes. Capacity $c(e)$ of grid edge e refers to the number of routing tracks that are allowed to legally cross the abutting boundary of two adjacent G-cells. The number of wires that pass through grid edge e is called the demand $d(e)$ of the grid edge. The overflow of a grid edge e is defined as the amount of demand in excess of capacity.


NCTU-GR 2.0
3D Global Routing Result
History cost information
3D Post Routing adopts maze routing to reroute every net using routing cost formulation in Eq. (6.1)
Negotiation-based Layer Assignment Algorithm
No
Iterations > R
New 3D result and $h_{NLA}(e_{i,z})$
Yes
Better 3D Global Routing Result

(a)



(b)

Fig. 6.3. (a) Design flow of Post3DGR. (b) Design flow of NLA

6.3 Design Flow of Post3DGR

Post3DGR performs the 3D post routing and negotiation-based layer assignment (NLA) iteratively on a given 3D global routing result to further minimize the result's wirelength, via count and overflows. Post3DGR can attach to the end of any global router to do optimization, this work performs Post3DGR following NCTU-GR 2.0.

Figure 6.3(a) shows the design flow of using Post3DGR to optimize the global routing result obtained by NCTU-GR 2.0, Post3DGR iterates until reaching a user defined iteration limitation *R*. The 3D post routing stage reroutes nets on 3D graph to reduce its wirelength and congestion, and then NLA re-arranges the routing layer for each wire to reduce vias. If it is necessary, NLA also can be extended to consider the issues such as antenna effect, timing, and double patterning. The details of NLA will be introduced in the next section.

Both 3D post routing and NLA can refine 3D global routing results. However, when 3D post routing and NLA are used individually to optimize the routing result, the improved quality is inclined to fast fall into sub-optimality. Although capable of altering routing topology, 3D post routing need to avoid passing through the grid edges whose $d(e) \geq c(e)$ to prevent overflows increasing. In contrast, although a net can negotiate with other nets to acquire the routing resource on other routing layers, each net cannot change its routing topology during NLA.

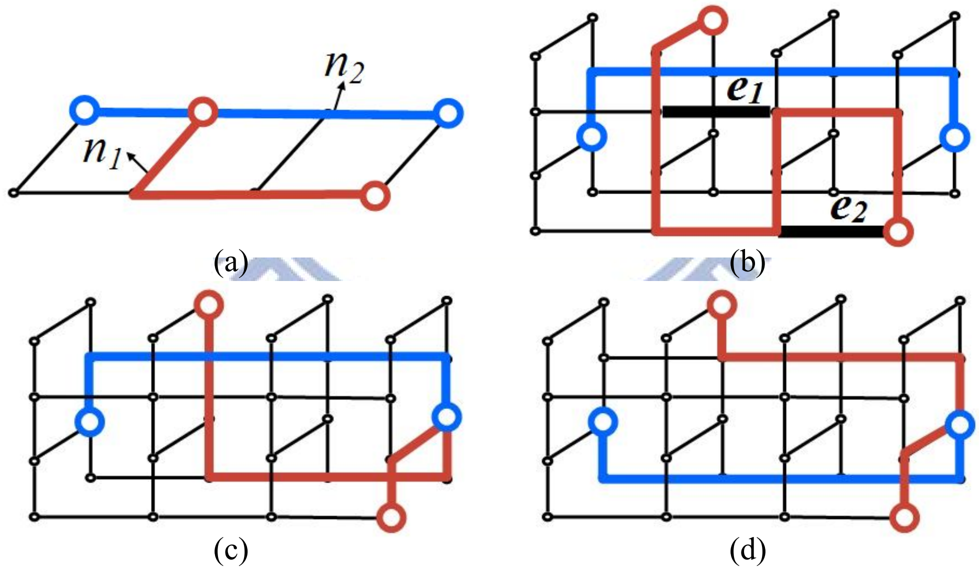


Fig. 6.4. Example of the quality improvement in Post3DGR. (a) A 2D routing result obtained by the 2D routing in NCTU-GR 2.0; (b) a 3D result obtained by the layer assignment in NCTU-GR 2.0; (c) the 3D result obtained by 3D post routing in Post3DGR; (d) the 3D result obtained by NLA in Post3DGR.

Through iterative topology change by 3D post routing and re-arranging routing layers by NLA, Post3DGR can refine a 3D routing results well. For example, a 2D routing result in Fig. 6.4(a) is obtained by the 2D routing stage of NCTU-GR 2.0, and then the layer assignment stage of NCTU-GR

2.0 map the 2D result to a 4-layer 3D grid graph to obtain a 3D routing result in Fig. 6.4(b). Since grid edges e_1 and e_2 cross obstacles, net n_1 needs a z-axis detour (via) to bypass obstacles. By assuming the length of a routing edge and a via both are one unit wirelength, and the capacity of each grid edge is one; the total wirelength of two paths in Fig. 6.4(b) is 15. To further optimize the 3D routing result, we feed the result in Fig. 6.4(b) into Post3DGR. The 3D post routing finds a shorter path for n_1 (Fig. 6.5(c)) with the reduced total wirelength to be 13. NLA then relaxes the resource at layer 3 used by n_2 for n_1 to further reduce the total wirelength to be 11 (Fig. 6.4(d)).

During the 3D post routing stage, every net is ripped-up and re-routed once by 3D maze routing. The 3D post routing stage adopts an ***inherited history cost*** function to guide the routing of each net, which can greedily reduce total wirelength. In NLA and the 2D routing stage of NCTU-GR 2.0, a grid edge with a high history cost implies that this grid edge frequently overflows and many nets desire to pass through this grid edge, so the routing resource of this grid edge is critical. The history cost information acquired by 2D routing and NLA is delivered to the 3D post routing in order to broaden the view of the routed nets in terms of knowing which grid edge is critical. The inherited history cost formulation is defined as follows:

$$\text{cost }(e_{i,z})=\begin{cases} C_1\times[d(e_{i,z})-c(e_{i,z})] & \text{if } d(e_{i,z})\geq c(e_{i,z}) \\ 1+C_2\times\left(\dfrac{d(e_{i,z})}{1+c(e_{i,z})}\right)^2+C_3\times hisC_I(e_{i,z}), & \text{otherwise} \end{cases} \qquad (6.1)$$

where $hisC_I(e_{i,z})$ denotes the inherited history cost of the 3D grid edge $e_{i,z}$; C_1, C_2 and C_3 are user defined constants and set as 10^5, 0.1 and 0.05 in this work, respectively. The great value of C_1 can reduce overflows or avoid overflow increasing. The $hisC_I(e_{i,z})$ is formulated as follows:

$$hisC_I(e_{i,z}) = NORM(h_{2D}(e_i))+NORM(h_{NLA}(e_{i,z})), \qquad (6.2)$$

where e_i is a 2D grid edge that is projected from $e_{i,z}$; $h_{NLA}(e_{i,z})$ denotes the history cost of $e_{i,z}$, as computed in previous NLA of the 3D post routing, and $h_{2D}(e_i)$ is the history cost from 2D routing of NCTU-GR 2.0. Since Post3DGR is designed to follow the execution of a 2D global router, $h_{2D}(e_i)$ is the final history cost of e_i, such as that defined in Eq. (5.6) as NRR of 2D routing ends. Because NLA follows the 3D post routing, the first round of the 3D post routing has no $h_{NLA}(e_{i,z})$ information. Thus, $NORM(h_{NLA}(e_{i,z}))$ is ignored in the first round of the 3D post routing. Function *NORM* normalizes $h_{2D}(e_i)$ and $h_{NLA}(e_{i,z})$ between zero to one. Function $NORM(h_{2D}(e_i))$ derives the normalized values as follows: The grid edges in the 2D grid graph with non-zero history costs are sorted in a non-decreasing order of their history costs. By assuming that the length of the ordering sequence is L_{2D}, $h_{2D}(e_i)$ is normalized to a/L_{2D} as e_i is the a-th element in this sorting sequence. Similarly, $NORM(h_{NLA}(e_{i,z}))$ is computed in the same manner.

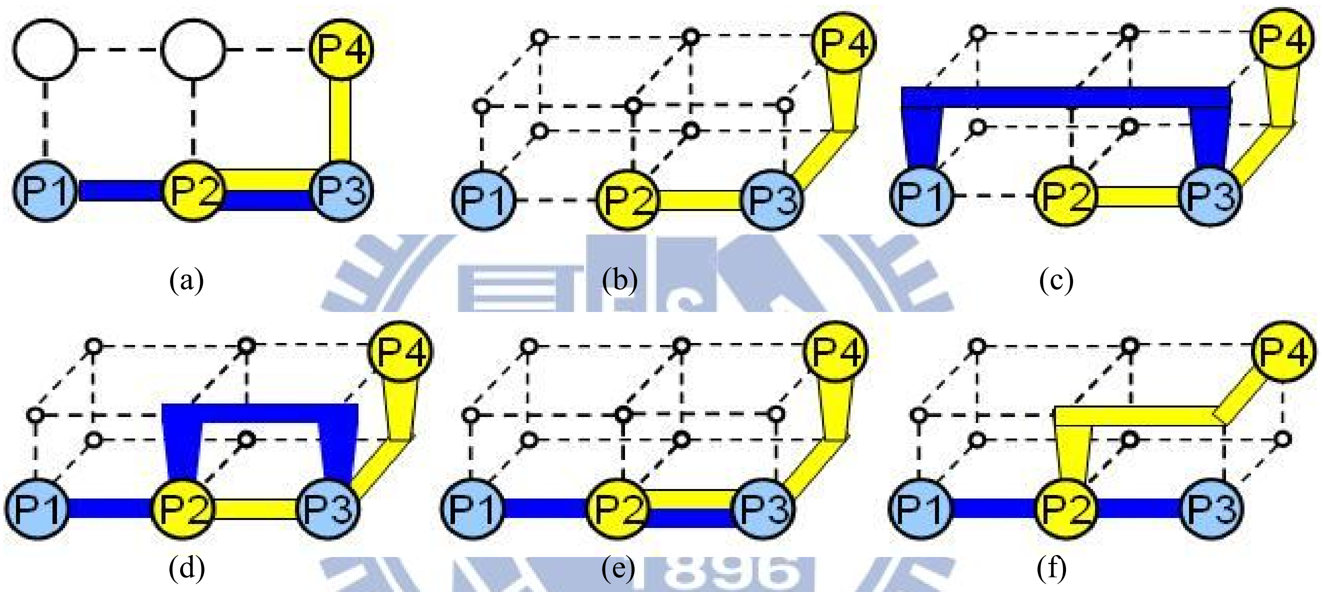


Fig. 6.5. The comparison between existing layer assignments and NLA. (a) Given a 2D routing, net *A* connects pins *p2* and *p4*, net *B* connects pins *p1* and *p3*; (b) assignment result of net *A*; (c) and (d) existing layer assignment results; (e) NLA result after the first iteration; (f) NLA result after the second iteration.

6.4 Negotiation-based Layer Assignment (NLA)

NLA is the kernel stage of Post3DGR, which focuses on minimizing vias without increasing overflows and without changing routing topology. Previous layer assignment works [16, 17, 27] addressing the via count minimization problem determined the assignment order of each net first and, then, adopted the dynamic-program-based layer assignment to assign each net sequentially. In order to avoid increasing overflows, the nets in the later assigning order have less available layer resources than those in the early assigning order. In this sorting-and-then-assigning method, the net ordering strongly impacts the result's quality. NLA can overcome the available resource problem for the nets in the later assigning ordering. In NLA, the later assigned nets can legally use their desired grid edges even whose routing resources are exhausted by the early assigned nets. NLA would iteratively rip-up and re-assign the overflowed nets and gradually increase the penalty of the overflowed grid edges. Because each net is assigned by a minimum-cost single net layer assignment (MCSNLA) algorithm, a net would gradually abandon their desired grid edges when the grid edge has high penalty. By adopting this negotiation scheme, the routing resource of a grid edge would be used by the nets who most need. NLA can reduce the impact of net ordering, and thus get a result with fewer vias than the work in [17, 27].

Figure 6.5 compares the layer assignment results with and without the proposed negotiation-based scheme. Figure 6.5(a) shows a 2D routing graph with two nets: net *A* connects pins *p2* and *p4* and net *B* connects pins *p1* and *p3.* By assuming that the capacity of each 3D grid edge is only one and the routing order of *A* is earlier than *B,* Fig. 6.5(b) shows the results of assigning net *A* on a 2-layer 3D graph. This example ignores the preferred routing direction rule for simplicity. Figures 6.5(c) and (d) show traditionally optimal results of assigning net *B*, each of which contain three vias. In contrast, NLA can first produce an overflowed result (Fig. 6.5(e)). Then, the penalty of the grid edge between *p2* and *p3* increases, and nets *A* and *B* are re-assigned sequentially to obtain the solution with only one via (Fig. 6.5(f)).

6.4.1 Algorithm Flow of NLA

Figure 6.3(b) shows the algorithm flow of NLA that comprises three stages. The initial layer assignment stage first rips-up every net from the 3D grid graph and then identifies a minimal via assignment solution for each net without addressing the congestion issue. Next, overflows produced in the initial layer assignment are minimized by the overflow reduction stage, in which the overflowed nets are iteratively re-assigned. If the layer assignment result has overflows, the congestion cost of the overflowed grid edges increases, and the overflowed nets are re-assigned until either a overflow-free result is obtained or overflows cannot be reduced anymore. Finally, the post optimization stage rips-up and re-assigns each net once to greedily further reduce via count but avoid increasing overflows. MCSNLA is adopted in all of initial layer assignment, overflow reduction, and post optimization stages to identify a minimum-cost layer assignment solution for a single net. The objective cost of MCSNLA consists of congestion cost and via cost, but the formulations of the congestion cost in different stages are different. The following subsections would introduce the algorithm of MCSNLA and the congestion cost formulations in each stage.

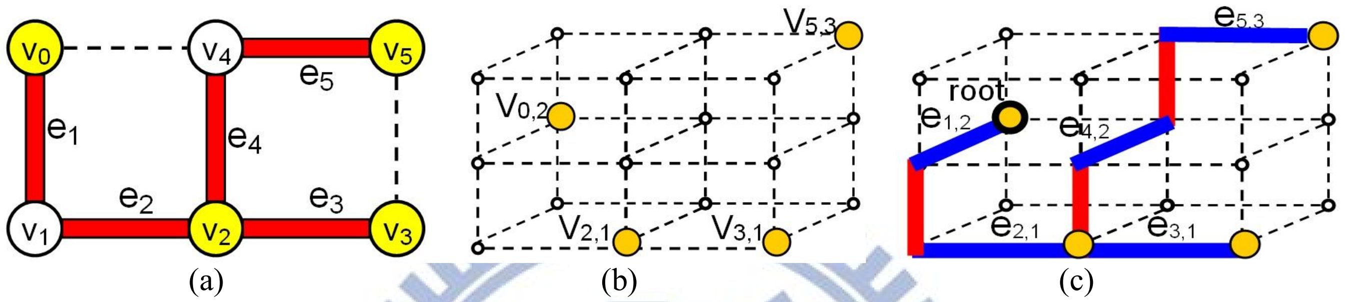


Fig. 6.6. An example of single net layer assignment. (a) a 2D routing topology of a net $N(V_{2D}, E_{2D})$; (b) a set of N's pin locations in G^k; (c) a layer assignment solution.

6.4.2 MCSNLA: Minimum-cost Single Net Layer Assignment

Figure 6.6 illustrates the single-net layer assignment problem. Given a 2D routing topology of a net $N(V_{2D}, E_{2D})$ where V_{2D} and E_{2D} denote the sets of 2D G-cells and 2D grid edges passed by N, respectively; and given a set of pin locations of N in G^k, which is denoted by P_{3D}. The single-net layer assignment problem for net N is to identify a 3D tree connecting all pins in P_{3D}, which consists of a set of 3D grid edges denoted E_{3D}, the edges of E_{3D} are the corresponding edges of E_{2D}. For instance, Given a 2D routing topology of $N(V_{2D}, E_{2D})$ in Fig. 6.6(a) and given P_{3D} in Fig. 6.6(b), where $V_{2D}=\{v_0, v_1, v_2, v_3, v_4, v_5\}$, $E_{2D}=\{e_1, e_2, e_3, e_4, e_5\}$ and $P_{3D}=\{v_{0,2}, v_{2,1}, v_{3,1}, v_{5,3}\}$, the single net layer assignment identifies a 3D tree as shown in Fig. 6.6(c), $E_{3D}=\{e_{1,2}, e_{2,1}, e_{3,1}, e_{4,2}, e_{5,3}\}$, this result contains three vias.

The algorithm flow of MCSNLA is similar to the single net layer assignment algorithm in [27], MCSNLA and the algorithm in [27] both adopt a dynamic programming technique. However, previous works is for via count minimization while this work addresses on minimizing the sum of the congestion cost and via cost. Before detail the algorithm of MCSNLA, we first introduce some notations.

1. k: the number of layers in the 3D grid graph.
2. ch(v_i): the set of child nodes of v_i. In Fig. 6.6(a), ch(v_2)={v_3, v_4}.
3. ch_e(v_i): the set of grid edges connecting v_i to its child nodes. In Fig. 6.6(a), ch_e(v_2)={e_3, e_4}.
4. pinL(v_i, P_{3D}): the layer of the pin at v_i. In Fig. 6.6(b), pinL(v_5)=3.
5. $T(v_i)$: a 2D tree rooted at v_i.
6. $t_{i,z}$: a 3D tree rooted at $v_{i,z}$. The $t_{i,z}$ is an assignment solution of $T(v_i)$. A 3D tree consists of a set of 3D grid edges and a set of required vias for connecting those grid edges and pins. Figure 6.6(c) shows a 3D tree $t_{0,2}$.
7. $S(v_{i,z})$: the set of 3D trees rooted at $v_{i,z}$.
8. $S(v_i)$: the set of assignment solutions of $T(v_i)$.
9. numVia($t_{i,z}$): the total via number of $t_{i,z}$. In Fig. 6.6(c), numVia($t_{0,2}$) is three.
10. via(Δ): Δ is a set of layers, via(Δ) denotes a set of required vias connecting the layers of Δ.

The objective cost function of MCSNLA is formulated as follows, MCSNLA can always identify the minimum-cost assignment solution .

$$\text{cost}(t_{i,z}) = viaCost \times \text{numVia}(t_{i,z}) + \sum_{e \in t_{i,z}} \text{congCost}(e), \quad (6.3)$$

where $t_{i,z}$ is a 3D tree identified by MCSNLA, *congCost*(e) denotes the congestion cost of grid edge e, and *viaCost* is an user defined constant for the cost of a single via. The formulation of *congCost*(e) will be detailed in the next subsection and *viaCost* is set to *100* in our implement. MCSNLA is a 2-phase algorithm based on dynamic-programming technique. During the first phase, the net $N(V_{2D}, E_{2D})$ is regarded as a 2D tree $T(v_0)$, v_0 is the root, the root and the leaves of $T(v_0)$ must contain a pin. Next, MCSNLA enumerates all possible 3D trees starting at leaf nodes in a bottom-up manner until reaching the root, MCSNLA would dynamically discard *inferior tree* (discarded later) to keep the size of solution space reasonable. In the second phase, the minimum-cost assignment solution for the entire tree is extracted from the set of possible 3D trees, then each net edge is assigned to the corresponding layer based on the minimum-cost assignment solution in a top-down manner. Figure 6.7 shows the pseudo code of MCSNLA. Lines 2-13 are the bottom-up phase, the procedure **InitSol** initializes a 3D tree rooted at the leaf node. The procedure **EnumSol** enumerates all possible 3D tree rooted at the internal node $v_{i,z}$. The procedure **PruneSol** discards the redundant trees from $S(v_{i,z})$ that the size of $S(v_{i,z})$ is limited in a reasonable range. Line 15 to line 18 is the top-down phase. Line 16 enumerates all possible assignment solutions for entire 3D tree into $S(v_{0,x})$. Line 17 extracts the minimum-cost one from $S(v_{0,x})$. Finally, each net edge of N is assigned to a corresponding layer of G^k according to the minimum-cost assignment solution.

```
Algorithm MCSNLA
Input:  net $N(V_{2D}, E_{2D})$, pin locations $P_{3D}$, 3D grid graph $G^k$;
1.  //Bottom-Up phase
2.  foreach node $v_i$ in the order given by a postorder traversal of $V_{2D}$,   $v_i$ is not the root.
3.          foreach layer $z$ from 1 to $k$
4.                  set $S(v_{i,z})$ to $\Phi$
5.                  if   $v_i$ is a leaf node
6.                          $S(v_{i,z})$←InitSol($v_{i,z}$, $P_{3D}$)
7.                  else
8.                          $S(v_{i,z})$←EnumSol($v_{i.z}$, ch($v_i$), ch_e($v_i$) , $P_{3D}$)
9.                  end if
10.                 PruneSol($S(v_{i,z})$)
11.         end foreach
12.         $S(v_i)=S(v_{i,0})\cup S(v_{i,1})\cup\ldots\cup S(v_{i,k})$
13. end foreach
14. //Top-Down phase
15. $x$=pinL($v_0$, $P_{3D}$)
16. $S(v_{0,x})$←EnumSol($v_{0.x}$, ch($v_0$), ch_e($v_0$), $P_{3D}$)
17. $t_{0,x}$←Select_solution($S(v_{0,x})$)
18. TopDown_Assignment( $N$, $G^k$, $t_{0,x}$ )
19. end
```

Fig. 6.7. The pseudo code of MCSNLA

Figure 6.8 shows the pseudo code of **InitSol**. At line 3, $|I|$ represents the number of required vias connecting the pin's layer to layer z. Because $t_{i,z}$ only includes vias, the cost of $t_{i,z}$ is only contains the via cost. Figure 6.9 shows the pseudo code of **EnumSol**. Without the loss of generality, we suppose that v_i has three child nodes in the pseudo code. At line 3, an 3D tree $t_{i,z}$ is constructed by composing $t_{a,la}$, $t_{b,lb}$,

$t_{c,lc}$, $e_{a,la}$, $e_{b,lb}$, $e_{c,lc}$ and I. The $t_{a,la}$, $t_{b,lb}$ and $t_{c,lc}$ are the sub-trees of $t_{i,z}$, the $e_{a,la}$, $e_{b,lb}$ and $e_{c,lc}$ are the 3D grid edges connecting the roots of $t_{a,la}$, $t_{b,lb}$ and $t_{c,lc}$ to the 3D nodes $v_{i,la}$, $v_{i,lb}$ and $v_{i,lc}$, respectively. The vias in I connect $v_{i,la}$, $v_{i,lb}$, $v_{i,lc}$ and $v_{i,z}$. If a pin is located at a layer of v_i, vias also need to connect to this pin. Figure 6.10 shows an example for constructing a 3D tree $t_{i,3}$ that consists of $t_{a,2}$, $t_{b,3}$, $t_{c,4}$, $e_{a,2}$, $e_{b,3}$, $e_{c,4}$, and I. At line 4, the cost of $t_{i,z}$ is identified by adding up the cost of $t_{a,la}$, $t_{b,lb}$, $t_{c,lc}$, the congestion cost of $e_{a,la}$, $e_{b,lb}$, $e_{c,lc}$, and the via cost of I. The loop of lines 1-6 enumerates all possible 3D trees rooted at $v_{i,z}$ and calculates the cost of each possible 3D tree. For instance, as the enumeration of all possible 3D trees rooted at $v_{2,2}$ for the net in Fig. 6.6, by assuming that $S(v_3)=\{t_{3,1}, t_{3,2}, t_{3,3}\}$, $S(v_4)=\{t_{4,1}, t_{4,2}, t_{4,3}\}$ and *viaCost* is set to 1, Fig. 6.11(a) shows the costs of sub-trees and the congestion costs of the 3D grid edges connecting the roots of sub-trees to $v_{2,2}$. Figure 6.11(b) shows all enumerated 3D trees rooted at $v_{2,2}$, and lists the cost of each enumerated 3D tree.

```
Procedure InitSol
Input:  node $v_{i,z}$, 3D pins location $P_{3D}$
1.  I= via( pinL($v_i$, $P_{3D}$), $z$)
2.  $t_{i,z}$=I
3.  cost($t_{i,z}$)=|I| *$viaCost$
4.  inset $t_{i,z}$ into $S(v_{i,z})$
5.  return $S(v_{i,z})$
```

Fig. 6.8. Procedure InitSol of MCSNLA

```
Procedure EnumSol
Input:  node $v_{i,z}$, ch($v_i$)=$\{v_a, v_b, v_c\}$, ch_e($v_i$)=$\{e_a, e_b, e_c\}$, pins location $P_{3D}$
1.  foreach solutions $t_{a,la}$, $t_{b,lb}$ and $t_{c,lc}$ of $S(v_a)$, $S(v_b)$ and $S(v_c)$, respectively.
2.          I= via($la$, $lb$, $lc$, pinL($v_i$, $P_{3D}$), $z$)
3.          $t_{i,z}$= $t_{a,la}$+$t_{b,lb}$+ $t_{c,lc}$+$e_{a,la}$+$e_{b,lb}$+ $e_{c,lc}$+I
4.          cost($t_{i,z}$)=cost($t_{a,la}$)+cost($t_{b,lb}$)+cost($t_{c,lc}$)+congCost($e_{a,la}$)+congCost($e_{b,lb}$)+
                    congCost($e_{c,lc}$)+ |I|*$viaCost$
5.          inset $t_{i,z}$ into $S(v_{i,z})$
6.  end foreach
7.  return $S(v_{i,z})$
```

Fig. 6.9. Procedure EnumSol of MCSNLA

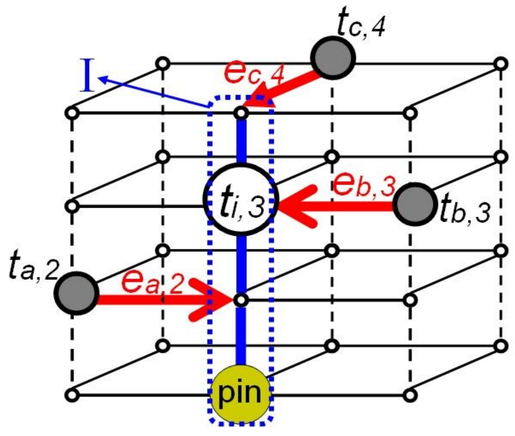


Fig. 6.10. An example for constructing a 3D tree $t_{i,3}$ that consists of $t_{a,2}$, $t_{b,3}$, $t_{c,4}$, $e_{a,2}$, $e_{b,3}$, $e_{c,4}$, and I

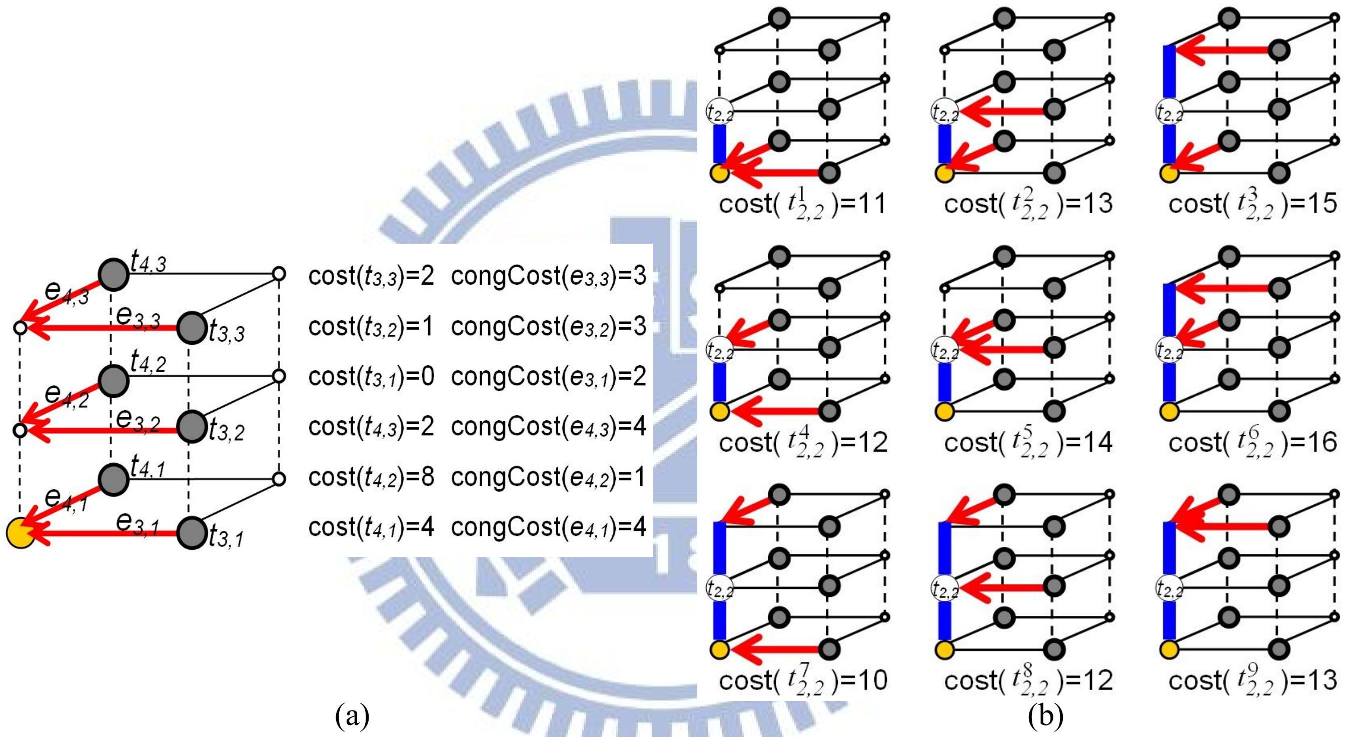


Fig. 6.11. An example of EnumSol. (a) The costs of 3D sub-trees and the congestion costs of the grid edges connecting to the root of each sub-tree to $v_{2,2}$; (b) all possible combinations to build tree $t_{2.2}$.

In order to limit the size of $S(v_{i,z})$ in a reasonable range, **PruneSol** discards the *inferior trees* from $S(v_{i,z})$. An inferior tree is defined as follows:

Definition 1. *inferior tree*: An 3D tree $t^w_{i,z}$ is regarded as an *inferior tree* if another 3D tree $t^u_{i,z}$ exists such that the cost($t^w_{i,z}$) is greater than cost($t^u_{i,z}$).

If there are more than one 3D trees in $S(v_{i,z})$, after pruning inferior trees, only the minimum-cost tree will be reserved in $S(v_{i,z})$, the other 3D trees will be discarded. For example, there are nine 3D trees enumerated by **EnumSol** in Fig. 6.11. After pruning inferior trees, only $t^7_{2,2}$ will be reserved in $S(v_{i,z})$.

Applying this scheme to pruning solutions can make MCSNLA obtain the minimum-cost assignment solution in a polynomial time. The time complexity of MCSNLA and the size of solution set are analyzed as follows.

Lemma 1. *For each node v_i, $i \neq 0$, the size of $S(v_i)$ is k.*

Proof: According to line 12 in Fig. 6.7, the size of $S(v_i)$ is the sum of the size of $S(v_{i,z})$, $1 \leq z \leq k$. Because only the minimum-cost tree is reserved in $S(v_{i,z})$, the size of $S(v_{i,z})$ is one. Thus, the size of $S(v_i)$ is k.

Lemma 2. *The time complexity of* **InitSol** *is $O(1)$. Moreover, as the child number of v_i is q, the time complexities of* **EnumSol** *and* **PruneSol** *for $v_{i,z}$, $1 \leq z \leq k$, are both $O(k^q)$.*

Proof: There are no loops in **InitSol** and all operations in **InitSol** are constant time, so the time complexity of **InitSol** is obviously $O(1)$. In **EnumSol**, the loop from line 1 to line 6 of Fig. 6.9 enumerates all combinations of the child node's solutions. With lemma 1, the solution set size of each child node is k that there are k^q combinations of the child node's solutions. Therefore, the loop of lines 1-6 runs k^q iterations, so the time complexity of **EnumSol** is $O(k^q)$. Moreover, **PruneSol** scans all solutions in $S(v_{i,z})$ to reserve the minimum-cost one. Because $S(v_{i,z})$ contains most k^q solutions enumerated by **EnumSol**, the time complexity of scanning k^q solutions in **PruneSol** is also $O(k^q)$.

Theorem 1. *The worst time complexity of MCSNLA is $O(|V_{2D}|*k^4)$, $|V_{2D}|$ represents the number of G-cells in 2D grid graph passed by N.*

Proof: In the bottom-up phase of Fig. 6.7, an inner loop is from line 3 to line 11 and an outer loop is from line 2 to line 13. **EnumSol** and **PruneSol** are the most time consuming parts of the inner loop, according to lemma 2, the time complexities of **EnumSol** and **PruneSol** are both $O(k^q)$, and the inner loop runs k iterations, so the time complexity of the inner loop is $O(k*(k^q+k^q))=O(k^{q+1})$. Moreover, the outer loop runs $|V_{2D}|$-1 iterations, thus the time complexity of the outer loop is $O((|V_{2D}|-1)*k^{q+1})=O(|V_{2D}|*k^{q+1})$. Because the child number q of each node except the root must be less or equal to three, the worst time complexity of the outer loop is $O(|V_{2D}|*k^4)$ as q is three. In the top-down phase, the worst time complexities of **EnumSol** and **Select_solution** are both O(k^4) since the child number of the root is four at most, and TopDown_Assignment takes the time of $|V_{2D}|$-1 to assign each net edge to the corresponding layer. Thus, the worst time complexity of whole MCSNLA is O($|V_{2D}|*k^4$)+O(k^4)+ O($|V_{2D}|$-1)=O($|V_{2D}|*k^4$).

Theorem 2. *MCSNLA can always identify the minimum-cost assignment solution.*

Proof: We prove this theorem by the contradiction method. By assuming that $t_{0,z}$ is the minimum-cost solution for a net but MCSNLA identifies a suboptimal solution $t'_{0,z}$. Because the cost of $t'_{0,z}$ is greater than $t_{0,z}$, following equation holds,

$$\begin{array}{l}\exists t'_{i,r} \text{ and } \exists t_{i,r},\ t'_{i,r} \subseteq t'_{0,z} \text{ and } t_{i,r} \subseteq t_{0,z},\ 1 \le r \le k. \\ \text{such that } \text{cost}(t'_{i,r}) > \text{cost}(t_{i,r})\end{array}, \qquad (6.4)$$

where $t'_{i,r}$ and $t_{i,r}$ are the sub-trees of $t'_{0,z}$ and $t_{0,z}$, respectively. The cost of $t'_{i,r}$ is greater than $t_{i,r}$. However, according to Definition 1, $t'_{i,r}$ is an inferior tree that will be discarded from the solution set. Because $t'_{i,r}$ is not reserved in the solution set, $t'_{0,z}$ is impossibly constructed by its sub-tree $t'_{i,r}$. This implies that MCSNLA never identifies a suboptimal solution $t'_{0,z}$. Namely, MCSNLA can always identify the minimum-cost assignment solution.

6.4.3 Congestion Cost Formulations

NLA involves three stages: initial layer assignment, overflow reduction and post optimization stages. MCSNLA adopts different congestion cost formulae in different stages for different objectives. In the initial layer assignment stage, a minimal via count solution is first identified without considering congestions for every net by fixing congestion cost as zero in Eq. 6.3. In the overflow reduction stage, if a grid edge frequently overflows, its congestion cost continually increases. The formulation of congestion cost is defined as follows:

$$congCost(e) = p_e * (1 + (h_e)^2), \qquad (6.5)$$

where e denotes a 3D grid edge, p_e denotes congestion penalty, and h_e represents the history cost. The congestion penalty is defined as follows:

$$p_e = 1 + \frac{\alpha}{1 + exp^{\beta * (cap(e) - dem(e))}}, \tag{6.6}$$

where α and β are user defined constants, α is set to 10 and β is set to 0.3 in this work. The history cost h_e increases by one as grid edge e overflows, and h_e keeps unchanged if e does not overflow. The value of h_e in the k-th iteration can be expressed as,

$$h_e^{k+1} = \begin{cases} h_e^k + 1, & if\ e\ overflows \\ h_e^k & , otherwise \end{cases}. \tag{6.7}$$

Figure 6.12 displays the changes of wire overflow and via count in the overflow reduction stage. As the process iteration proceeds, the overflow decreases since the increasing congestion cost encourages MCSNLA to obtain the solution with less overflows at the cost of increasing via counts. In the post optimization stage, each net is re-assigned to greedily reduce vias but without increasing overflows. Equation (6.8) shows the congestion cost formula used in the post optimization stage, where σ refers to an extremely large constant.

$$congCost(e) = \begin{cases} \sigma * (1 + d(e) - c(e)), & if\ \ d(e) \geq c(e) \\ 0 & , otherwise \end{cases} \tag{6.8}$$

Equation (6.8) enlarges the penalty of an overflow assignment that MCSNLA identify minimum-overflow assignment solution. If at least an overflow-free assignment exists, MCSNLA based on Eq. (6.8) would identify the minimum via count assignment among the overflow-free assignments.

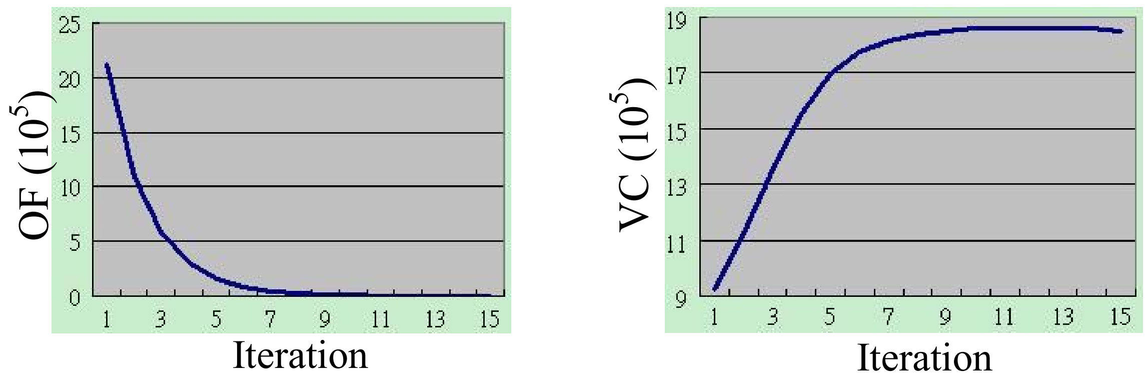


Fig. 6.12 The overflow reduction stage of NLA resolves overflow (OF) of adaptec2 at the cost of increasing vias.

6.5 Experimental Results

The proposed algorithms were implemented in C/C++ on a quad-core 2.4 GHz Intel Xeon-based linux server with 32GB memory. ISPD08 [2] benchmark circuits are used in the experiments. Each benchmark has either six or eight routing layers. The preferred direction of even layers is horizontal, while that of odd layers is vertical. This section consists of three parts. First, to compare NLA with other layer assignment works [17, 27] on via count minimization problem, each algorithm reads the same global routing results of ISPD08 benchmark, which are produced by the NTHU-Route 2.0 [11]. Different layer assignment algorithms then transform 2D routing results to 3D. Next, the effectiveness of Post3DGR including NLA is demonstrated. Finally, we extend NLA to consider antenna effect.

TABLE 6.1 COMPARING NLA WITH PREVIOUS LAYER ASSIGNMENT WORKS ON VIA COUNT MINIMIZATION PROBLEM.

Benchmark	Via Count (10^5)			Runtime		
	[27]	[17]	NLA	[27]*	[17]	NLA
adaptec1	17.69	17.03	16.69	27.5	20.73	48.1
adaptec2	19.3	18.57	18.31	25.67	21.21	42.16
adaptec3	34.91	33.56	32.9	82.5	57.72	126.81
adaptec4	32.15	31.21	30.82	75.17	51.14	105.54
adaptec5	52.4	50.35	49.3	99	87.82	136.74
newblue1	22.22	21.76	21.42	22.92	22.45	35
newblue2	29.46	28.61	28.14	39.42	34.46	56.1
newblue3	30.23	29.37	29	59.58	41.23	94.55
newblue4	47.05	45.89	44.73	79.75	55.92	100.68
newblue5	84.51	81.51	80.16	134.75	99.67	211.22
newblue6	74.66	72.64	71.01	105.42	84.42	143.53
newblue7	166.01	161.52	157.21	306.17	240.83	388.03
bigblue1	18.73	18.03	17.6	38.5	31.98	55.71
bigblue2	42.11	40.75	40.32	52.25	36.77	74.28
bigblue3	52.43	51.47	50.55	90.75	75.72	149.2
bigblue4	109.14	107.1	104.69	166.83	123.46	259.29
Ratio$_{ind}$	1.051	1.019	1	0.678	0.531	1
Ratio$_{sum}$	1.051	1.021	1	0.694	0.536	1

*AMD Dual Core Opteron Processor 2.2-GHz CPU

6.5.1 Effectiveness of NLA

Table 6.1 compares the proposed NLA with previous layer assignment works [17, 27] for via count minimization. The overflows are not listed in Table 6.1 since the results of each layer assignment algorithm have the same overflows. The first major column lists the total via count of the results of [27], [17] and NLA, in which NLA assigns nets in an increasing order of net IDs. This work using casual net ordering still can get the results with lower via count that [17, 27]. The second major column in Table 6.1 compares the runtimes of [27], [17] and NLA. Since the algorithm in [27] runs on a different platform from others, the runtime of [27] is normalized with the clock rate ratio. NLA consumes more runtime than [27, 17] because NLA iteratively re-assigns overflowed nets to explore a better solution.

TABLE 6.2 THE QUALITY VARIATIONS OF NLA AND [17] WITH DIFFERENT ASSIGNMENT ORDERING SEQUENCES.

Bench-mark	Total Via Count (10^5)									
	[17]					NLA				
	DW	IW	DP	IP	Vari	DW	IW	DP	IP	Vari
adaptec1	27.53	18.96	22.81	22.26	8.56	16.69	16.7	16.68	16.71	0.02
adaptec2	29.52	20.57	24.53	24.16	8.95	18.31	18.33	18.3	18.33	0.03
adaptec3	55.26	37.3	45.29	43.95	17.96	32.9	32.92	32.89	32.93	0.05
adaptec4	50.65	34.75	41.61	40.78	15.9	30.82	30.85	30.8	30.86	0.05
adaptec5	81.02	55.81	66.51	66.51	25.21	49.3	49.35	49.27	49.37	0.09
newblue1	33.26	23.41	27.94	26.97	9.86	21.42	21.43	21.41	21.44	0.03
newblue2	48.74	31.37	38.66	38.85	17.37	28.13	28.17	28.12	28.17	0.05
newblue3	49.35	32.41	38.61	40.11	16.94	28.98	29.01	28.97	29.02	0.05
newblue4	73.16	50.78	60.61	59.67	22.39	44.74	44.76	44.71	44.77	0.06
newblue5	136.5	98.18	107	103.86	38.32	80.16	80.35	80.12	80.27	0.23
newblue6	116.76	79.52	96	93.95	37.24	71.02	71.06	70.95	71.08	0.13
newblue7	267.23	176.88	213	210.02	90.35	157.21	157.31	157.15	157.38	0.23
bigblue1	28.2	19.53	23.25	23.62	8.67	17.6	17.61	17.59	17.61	0.02
bigblue2	64.46	46.19	52.41	52.14	18.27	40.31	40.36	40.3	40.36	0.06
bigblue3	80.25	56.78	69.36	66.28	23.48	50.54	50.58	50.51	50.61	0.09
bigblue4	175.39	118.04	143.29	144.42	57.35	104.69	104.8	104.57	104.78	0.23
Ratio	**1.65**	**1.13**	**1.35**	**1.33**	**322.86**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**

Table 6.2 shows the quality variations (Vari) of the work in [17] and NLA with different assignment ordering sequences. The comparison does not include the work in [27] since its source code and binary are unavailable. Four ordering sequences are adopted – decreasing wirelength (DW), increasing wirelength (IW), decreasing pin's number (DP) and increasing pin's number (IP). Table 6.2 reveals that the net ordering sequence easily impacts the quality of [17], and the proposed NLA yields lower variation in the total via count as different net assignment ordering sequences are applied.

TABLE 6.3 THE WIRELENGTH IMPROVEMENT AND RUNTIME OF POST3DGR.

ISPD'08 benchmark	Post3DGR		ISPD'08 benchmark	Post3DGR	
	WL(%)	CPU(m)		WL(%)	CPU(m)
adaptec1	2.66	8.82	newblue4	2.62	14.70
adaptec2	3.03	7.84	newblue5	2.96	28.42
adaptec3	2.47	36.26	newblue6	2.23	10.78
adaptec4	2.98	20.58	newblue7	2.89	41.16
adaptec5	2.44	18.62	bigblue1	2.31	6.86
newblue1	2.99	7.84	bigblue2	2.59	7.84
newblue2	3.49	9.80	bigblue3	3.01	21.56
newblue3	3.04	577.22	bigblue4	2.89	30.38

6.5.2 Effctiveness of Post3DGR

Table 6.3 shows the global routing wirelength improvement (WL%) and the runtime of using the proposed Post3DGR to further refine the 3D global routing results produced by NCTU-GR 2.0 (Tables 5.6 and 5.7). Post3DGR, which performs the 3D post routing and NLA twice, can achieve an improved wirelength of 2.79% on average. Compared to the runtime of NCTU-GR 2.0 in Table 5.6, the runtime of Post3DGR is enormous due to large 3D searching space. However, compared to the pure 3D global routers [3-5], the runtime of Post3DGR is negligible.

We name the flow of NCTU-GR 2.0 followed by Post3DGR to NCTU-3D-GR. Table 6.4 compares the routing results of NCTU-3D-GR with MGR [6] and GRIP [4]. MGR is a multi-level 3D router. GRIP [4] is a parallel ILP-based 3D router, which obtains the best wirelength in all open literature. Although PGRIP [5] enhances the parallelism degree of GRIP to improve the runtime, the wirelength reported in PGRIP is worse than that in GRIP. Given its objective to yield high-quality results, this work compares NCTU-3D-GR with GRIP rather than with PGRIP. In Table 6.4, TOF and WL denote total overflow and total global routing wirelength (including vias and edges), respectively. Table 6.4 shows that NCTU-3D-GR identifies 0.2% and 2.9% less wirelength than GRIP and MGR, respectively.

In Table 6.4, NCTU-3D-GR and MGR is performed on a single core. In contrast, GRIP was run on a heterogeneous grid of CPUs, shared by many users, and controlled by the Condor grid computing toolkit. In the major column of GRIP, the wall time (Wall) is the real time from GRIP launched to termination, while the total time (CPU) is the summation of the runtimes spent by all CPUs. Table 6.4 reveals that NCTU-3D-GR consumes a significantly lower runtime and computation power than those of GRIP, but the routing results of NCTU-3D-GR beat GRIP.

TABLE 6.4 COMPARING NCTU-3D-GR 2.0 WITH OTHER 3D ROUTERS.

	NCTU-3D-GR			GRIP [15]				MGR [17]		
	TOF	WL	CPU	TOF	WL	Wall	CPU	TOF	WL	CPU
adaptec1	0	51.63	11.34	0	51.33	388	2101	0	52.82	4.39
adaptec2	0	49.95	8.49	0	49.93	455	2704	0	51.46	1.04
adaptec3	0	126.05	39.66	0	126.8	478	6319	0	128.92	4.83
adaptec4	0	116.94	21.81	0	118.43	509	5221	0	119.96	1.41
adaptec5	0	150.25	24.65	0	149.5	584	3175	0	153.23	7.95
newblue1	0	44.39	10.21	0	44.57	483	2306	0	45.58	4.51
newblue2	0	71.91	10.44	0	72.47	467	4192	0	74.46	0.80
newblue3	31934	102.16	720.56	45960	102.83	1430	14590	31026	107.22	19.99
newblue4	142	123.94	32.03	136	124.4	529	2944	136	128.54	15.64
newblue5	0	221.91	33.68	0	222.8	821	4593	0	228	6.54
newblue6	0	171.58	15.22	0	170.5	448	2219	0	174.86	7.04
newblue7	54	332.99	126.83	54	335.8	985	4788	56	349.02	110.12
bigblue1	0	54.99	10.95	0	53.7	339	956	0	55.82	5.04
bigblue2	0	86.36	12.05	0	86	690	3411	0	88.92	6.00
bigblue3	0	125.46	23.47	0	126.2	731	2690	0	128.75	2.89
bigblue4	152	218.50	90.61	180	220.7	726	3096	134	225.73	21.31
Ratio		**1**			**1.002**				**1.029**	

6.5.3 Consideration of Antenna Effect

Generally speaking, the goal of NLA is to minimize the cost of an objective function. This work formulates congestion cost and via cost into the objective function (Eq. 6.3), so NLA can minimize via count and congestion. If we formulate manufacturing issues into the objective function of NLA, NLA also can be aware of these issues. Moreover, Post3DGR including NLA can consider these manufacturing issues too. We have extended NLA to consider antenna effect to develop an antenna-aware NLA [33] to. Please refer to our paper [33] to know the details of antenna-aware NLA [33].

TABLE 6.5 COMPARING ANTENNA-AWARE NLA WITH OTHER LAYER ASSIGNMENT ALGORITHMS.

Bench -mark	COLA [2]			NLA			LAVA [4]			Antenna-ware NLA		
	#vn	Vias (10^5)	Cpu* (min)	#vn	Vias (10^5)	Cpu (min)	#vn	Vias (10^5)	Cpu* (min)	#vn	Vias (10^5)	Cpu (min)
adaptec1	911	17.69	0.46	709	16.69	0.80	602	17.51	0.73	4	16.72	14.82
adaptec2	879	19.30	0.43	712	18.31	0.70	568	19.07	0.64	0	18.33	12.86
adaptec3	2959	34.91	1.38	2919	32.90	2.11	2194	34.58	1.94	5	33.00	60.03
adaptec4	2009	32.15	1.25	1925	30.82	1.76	1931	31.93	1.61	4	30.90	41.11
adaptec5	4166	52.40	1.65	3744	49.30	2.28	2465	51.9	2.09	0	49.43	53.53
newblue1	328	22.22	0.38	460	21.42	0.58	273	24.95	0.53	6	21.43	4.93
newblue2	681	29.46	0.66	534	28.14	0.94	444	29.15	0.86	1	28.18	11.79
newblue3	466	30.23	0.99	429	29.00	1.58	251	29.42	1.44	1	29.08	29.48
newblue4	874	47.05	1.33	849	44.73	1.68	617	46.59	1.54	0	44.77	20.23
newblue5	3009	84.51	2.25	2766	80.16	3.52	2137	83.79	3.23	0	80.30	77.93
newblue6	3453	74.66	1.76	3280	71.01	2.39	2736	73.83	2.19	0	71.12	33.27
newblue7	10286	166.01	5.10	8628	157.21	6.47	5844	164.52	5.93	0	157.50	354.41
bigblue1	1841	18.73	0.64	1459	17.60	0.93	1423	18.57	0.85	0	17.65	19.71
bigblue2	392	42.11	0.87	389	40.32	1.24	264	41.72	1.13	0	40.34	16.25
bigblue3	3576	52.43	1.51	3631	50.55	2.49	2692	51.99	2.28	0	50.66	233.01
bigblue4	7676	109.14	2.78	8627	104.69	4.32	5230	108.28	3.96	0	104.93	301.91
Sum	**43506**			**41261**			**29671**			**21**		
Ratio		**1.049**	**0.036**		**0.998**	**0.053**		**1.046**	**0.123**		**1**	**1**

*AMD Dual Core Opteron Processor 2.2-GHz CPU

Table 6.5 compares the proposed antenna-aware NLA with the other layer assignment works, in which COLA [27] and NLA focus on the via count minimization but does not consider the antenna effect, and LAVA

[32] is an antenna avoidance layer assignment. Each algorithm reads the global routing results produced by NTHU-Route 2.0, and then re-assigns the layers to every net edge. NLA and antenna-aware NLA perform on our machine. The routing results of COLA and LAVA are quoted from [32]. Because the performing machines of COLA and LAVA are different than NLA and antenna-aware NLA, the runtimes of COLA and LAVA are normalized by the clock rate. The number of overflows is not listed in Table 6.5 since the results of all layer assignment algorithms have the same number of overflows. Table 6.5 reveals that antenna-aware NLA can effectively reduce the number of nets with antenna violations (#vn). In 16 benchmarks, antenna-aware NLA can yield 10 antenna-violation-free results while the other works yield no antenna-violation-free result. As for the total number of antenna violations in all benchmarks, antenna-aware NLA, COLA, NLA and LAVA yield 21, 43506, 41261 and 29671 antenna violations, respectively. In addition, the via count of antenna-aware NLA is less than COLA and LAVA by 4.9% and 4.6%, respectively.

6.6 Summary

The proposed Post3DGR consists of NLA stage and the post 3D routing stage. Given a 3D global routing result, Post3DGR coordinates these two stage to reduce the via count, wirelength and congestion of the 3D result. Particularly, NLA can overcome the available resource problem, and the later assigned nets can legally compete with the early assigned nets for their desired routing resource to obtain better results. Experiments show that NCTU-GR 2.0 with Post3DGR identifies the best routing results on ISPD benchmarks with much less time than GRIP.

Chapter 7 Conclusions and Future Works

7.1 Conclusions

This dissertation presents a routability-driven placement and routing (P&R) flow based on the proposed global routing engines. In the different stages of P&R flow, global routing engines play different roles to guide the flow. At first, we develop a fast global-routing-based RCE tool called Grace that can cooperates with placers to improve the routability of placement solutions. Grace invokes unilateral monotonic routing and HUM routing to replace time-consuming maze routing, and adopts a congestion-aware bounding box expansion method to efficiently get a global routing result. Moreover, to fulfill industrial demands, we enhance Grace to consider local congestion, scenic controlling, layer directive constraint and target congestion ratio. The enhanced Grace is practically used in the industrial flow.

Grace can generate a congestion map and then guide placers to improve the routability of placement solutions, but the congestion map changes as cells are moved by subsequent placement operations, lowering the accuracy of congestion estimation and then the routability of a placement. To eliminate this mismatch, we develop a routability optimizer Ropt that takes a placement solution and optimizes its routability by incremental place-and-route. The innovations of Ropt include local-routability-aware global routing model, routing-cost-driven global re-placement, legalization with global routing preserved, and local detailed placement.

Given a placement solution, NCTU-GR 2.0 adopts the proposed bounded-length maze routing algorithms, RSMT-aware routing scheme, and dynamically adjusted history cost function to better the usage of routing resource and thus obtain a high-quality 3D global routing result. Then, Post3DGR further optimizes the 3D global routing result to reduce its wirelength, via count and congestion. Also, Post3DGR can be extended to consider the manufacturing issues such as antenna effect, which can ease the runtime and efforts of downstream detailed routers.

7.2 Future Works

As the design complexity continues to increase, the demands to bridge the gap between placement and routing grow rapidly. To complete the routability- and performance-aware placement and routing flow presented in this dissertation (Fig. 1.2), a routability-aware placer based on the proposed global routing engine is compulsory. Also, I want to learn more about detailed routing in industry, and then try to make contributions to the development of next-generation detailed routers. Although it is very difficult, my ultimate target is to develop a commercial router and then present a highly-unified place-and-route tool.

In Chapter 3, even when pin density and blockage effect are formulated into global routing model, global-routing-based RCE is still not accurate enough in predicting detailed routability. To more precisely estimate detailed routability, the development of a track-routing-based RCE to cooperate with global-routing-based RCE becomes compulsory. Using global-routing-based and track-routing-based RCEs together can get both global and local congestion information for estimating detailed routability. Moreover, I plan to embed the utility of track routing into NCTU-GR 2.0 such that NCTU-GR 2.0 can plan global routing path and routing tracks for each net simultaneously.

Bibliography

[1] ISPD07 contest: http://archive.sigda.org/ispd2007/contest.html

[2] ISPD08 contest: http://archive.sigda.org/ispd2008/contests/ispd08rc.html

[3] J. A. Roy and I. L. Markov, "High-performance routing at the nanometer scale," *IEEE TCAD*, 27(6), pp. 1066-1077, 2008.

[4] T.-H. Wu et al, "GRIP: scalable 3-D global routing using integer programming," *IEEE TCAD*, 30(1), pp. 72-84, 2011.

[5] T.-H. Wu et al, "A parallel integer programming approach to global routing," in *Proc. DAC*, pp. 194-199, 2010.

[6] Y. Xu and C. Chu, "MGR: multi-level global router," in *Proc. ICCAD*, 2011.

[7] J. Hu et al, "Completing high-quality global routes," in *Proc. ISPD*, 2010.

[8] M. Moffitt, "MAIZEROUTER: engineering an effective global router," *IEEE TCAD*, 27(11), pp. 2017-2026, 2008.

[9] M. Cho et al, "BoxRouter 2.0: architecture and implementation of a hybrid and robust global router," in *Proc. ICCAD*, 2007.

[10] M. M. Ozdal and M. D.F. Wong, "ARCHER: a history-driven global routing algorithm," in *Proc. ICCAD*, 2007.

[11] Y.-J. Chang et al, "NTHU-Route 2.0: a roubust global router for modern design," *IEEE TCAD*, 29(12), pp. 1931-1944, 2010.

[12] H.-Y. Chen et al. "High-performance global routing with fast overflow reduction." in *Proc. ASP-DAC*, 2009.

[13] M. Pan and C. Chu, "FastRoute : a step to integrate global routing into placement," in *Proc. ICCAD*, 2006.

[14] M. Pan and C. Chu, "FastRoute 2.0: a high-quality and efficient global router," in *Proc. ASP-DAC*, 2007.

[15] Y. Zhang et al, “FastRoute3.0: a fast and high quality global router based on virtual capacity,” in *Proc. ICCAD,* 2008.

[16] Y. Xu et al, “FastRoute 4.0: global router with efficient via minimization,” in *Proc. ASP-DAC,* 2009.

[17] K.-R. Dai et al, “NCTU-GR: efficient simulated evolution-based rerouting and congestion-relaxed layer assignment on 3-D global routing,” *IEEE TVLSI*, 20(3), pp. 459-472, 2012.

[18] W.-H. Liu et al, “Multi-threaded collision-aware global routing with bounded-length maze routing,” in *Proc. DAC*, 2010.

[19] Y. Han et al, “Exploring high throughput computing paradigm for global routing,” in *Proc. ICCAD*, 2011.

[20] W.-H. Liu et al, “High-quality global routing for multiple dynamic supply voltage designs,” in *Proc. ICCAD*, 2011.

[21] W.-H. Liu et al, “A fast maze-free routing congestion estimator with hybrid unilateral monotonic routing,” in *Proc. ICCAD*, 2012.

[22] L. McMurchie and C. Ebeling, “Pathfinder : a negotiation-based performance-driven router for FPGAs,” in *Proc. ACM Int’l Symp on Field-Programmable Gate Arrays*, 1995.

[23] C. Chu and Y.-C. Wong, “FLUTE: fast lookup table based rectilinear steiner minimal tree algorithm for VLSI design,” *IEEE TCAD*, 27(1), pp. 70-83, 2008.

[24] M. Garey and D. Johnson, “Computers and intractability: a guide to the theory of NP-completeness,” New York: W.H. Freeman and Co., 1979.

[25] H. C. Joksch, “The shortest route problem with constraints,” *Journal of Mathematical Analysis and Applications*, vol. 14, pp. 191–197, 1966.

[26] S. Chen et al, “Two techniques for fast computation of constrained shortest paths,” *IEEE/ACM Trans. on Networking*, 16(1), 2008.

[27] T.-H. Lee and T.-C. Wang, “Congestion-constrained layer assignment for via minimization in global routing,” *IEEE TCAD*, 27(9), pp. 1643-1656, 2008.

[28] W.-H. Liu, and Y.-L. Li, "Negotiation-based layer assignment for via count and via overflow minimization," in *Proc. ASP-DAC*, 2011.

[29] T.-H. Lee and T.-C. Wang, "Robust layer assignment for via optimization in multi-layer global routing," in *Proc. ISPD*, 2009.

[30] J. Sun et al, "Post-routing layer assignment for double patterning," in *Proc. ASP-DAC*, 2013.

[31] Z. Li et al, "Fast interconnect synthesis with layer assignment", in *Proc. ISPD*, 2008.

[32] T.-H. Lee and T.-C. Wang, "Simultaneous antenna avoidance and via optimization in layer assignment of multi-layer global routing," in *Proc. ICCAD*, 2010.

[33] W.-H. Liu and Y.-L. Li, "Optimizing antenna area and separators in layer assignment of multi-layer global routing," in *Proc. ISPD*, 2012.

[34] N. J. Naclerio et al, "The via minimization problem is NP-complete," *IEEE Trans. Computers*, 38(1), pp. 1604–1608, 1989.

[35] J. Lou et al, "Estimating routing congestion using probabilistic analysis," *IEEE TCAD*, 21(1), pp. 32–41, 2002.

[36] J. Westra et al, "Probabilistic congestion prediction," in *Proc. ISPD*, 2004.

[37] H. Shojaei et al, "Congestion analysis for global routing via integer programming," in *Proc. ICCAD*, 2011.

[38] X. He et al, "Ripple: an effective routability-driven placer by iterative cell movement," in *Proc. ICCAD*, 2011.

[39] M.-K. Hsu et al, "Routability-driven analytical placement for mixed-size circuit designs," in *Proc. ICCAD*, 2011.

[40] K. Tsota et al, "Guiding global placement with wire density," in *Proc. ICCAD*, 2008.

[41] J. A. Roy et al, "Seeing the forest and the trees: Steiner wirelength optimization in placement," *IEEE TCAD*, 26(4), pp. 632–644, 2007.

[42] M.-C. Kim et al, "A SimPLR method for routability-driven placement," in *Proc. ICCAD*, 2011.

[43] M. Pan and C. Chu, "IPR: an integrated placement and routing algorithm," in *Proc. DAC*, 2007.

[44] J. Roy et al, "CRISP: congestion reduction by iterated spreading during placement," in *Proc. ICCAD*, 2009.

[45] K.-R. Dai et al, "GRPlacer: improving routability and wire-length of global routing with circuit replacement," in *Proc. ICCAD*, 2009.

[46] N. Viswanathan et al, "The ISPD-2011 routability-driven placement contest and benchmark Suite," in *Proc. ISPD*, 2011.

[47] N. Viswanathan et al, "The DAC 2012 routability-driven placement contest and benchmark suite," in *Proc. DAC*, 2012.

[48] N. Viswanathan et al, "ICCAD-2012 CAD contest in design hierarchy aware routability-driven placement and benchmark suite," in *Proc. ICCAD*, 2012.

[49] M. D. Moffitt, "Global routing revisited," in *Proc. ICCAD*, 2009.

[50] Y.-J. Chang et al, "GLADE: a modern global router considering layer directives," in *Proc. ICCAD*, 2010.

[51] T.-H. Lee et al, "An enhanced global router with consideration of general layer directives," *in Proc. ISPD*, 2011.

[52] M. D. Moffitt and C. N. Sze, "Wire synthesizable global routing for timing closure," in *Proc. ASP-DAC,* 2011.

[53] Y. Wei et al, "GLARE: global and local wiring aware routability evaluation," in *Proc. DAC*, 2012.

[54] Y. Wei et al, "CATALYST: planning layer directives for effective design closure," in *Proc. DATE*, 2013.

[55] Hai Zhou et al, "Efficient minimum spanning tree construction without Delaunay triangulation," *Information Processing Letter*, 2002.

[56] T. Taghavi et al, "New placement prediction and mitigation techniques for local routing congestion," in *Proc. ICCAD,* 2010.

[57] J. Cong et al, "Optimizing routability in large-scale mixed-size placement," in *Proc. ASP-DAC*, 2013.

[58] Y. Zhang and C. Chu, "Fast and effective congestion refinement of placement," in *Proc. ICCAD,* 2009.

[59] http://www.cadence.com/products/di/soc_encounter/

[60] P. Spindler et al, "Abacus: fast legalization of standard cell circuits with minimal movement," in *Proc. ISPD*, 2008.

[61] Y.-M. Lee et al, "A hierarchical bin-based legalizer for standard-cell designs with minimal disturbance," in *Proc. ASP-DAC*, 2010.

[62] A. Agnihotri et al, "Fractional cut: improved recusive bisecction placement," in *Proc. ICCAD*, 2003.

[63] http://archive.sigda.org/dac2012/contest/dac2012_contest.html

[64] W.-H. Liu et al, "Case study for placement solutions in ISPD11 and DAC12 routablity-driven placement contests," in *Proc. ISPD*, 2013.

[65] W.-H. Liu et al, "Globally routing multiple dynamic supply voltage designs," in *Proc. VLSI Design/CAD Symposium*, 2011.

[66] Y.-H. Lin et al, "Topology-aware buffer insertion and GPU-based massively parallel rerouting for ECO timing optimization," in *Proc. ASP-DAC*, 2012.

[67] W.-H. Liu et al, "Minimizing clock latency range in robust clock tree synthesis," in *Proc. ASP-DAC*, 2010.

[68] L.-C. Chou et al, "ECO timing optimization with elmore delay model," in *Proc. ITC-CSCC*, 2011.

[69] J.-R. Gao et al, "A new global router for modern designs," in *Proc. ASP-DAC*, 2008.

[70] M. Cho and D. Z. Pan, "Boxrouter: A new global router based on box expansion and progressive ILP," *IEEE TCAD*, 26(12), pp. 2130–2143, 2007.

[71] W.-H. Liu et al, "NCTU-GR 2.0: Multi-Threaded Collision-Aware Global Routing with Bounded-Length Maze Routing," *IEEE TCAD*, accept.