

# 國立交通大學

資訊科學系

碩士論文

有效率的系統晶片後端設計變更方塊式繞  
線器



An Efficient Tile-Based ECO Router for SoC Designs

研究生：李建毅

指導教授：李毅郎 博士

中華民國九十三年十月

有效率的系統晶片後端設計變更方塊式繞線器  
An Efficient Tile-Based ECO Router for SoC Designs

研 究 生：李建毅

Student : Jian-Yin Li

指導教授：李毅郎

Advisor : Dr. Yih-Lang Li

國立交通大學  
資訊科學研究所  
碩士論文



A Thesis  
Submitted to Institute of Computer and Information Science  
College of Electrical Engineering and Computer Science  
National Chiao Tung University  
in partial Fulfillment of the Requirements  
for the Degree of  
Master  
in

Computer and Information Science

Oct 2004

Hsinchu, Taiwan, Republic of China

中華民國九十三年十月

# 有效率的系統晶片後端設計變更方塊式繞線器

學生：李建毅

指導教授：李毅郎 博士

國立交通大學 資訊科學系 碩士班

## 摘 要

由於製程與電路設計技術顯著的進展，設計複雜度達到數百萬閘級，而系統單晶片的設計需求更使得晶片複雜度提高與不同設計種類如混合訊號與高頻電路佈局的整合複雜度提高。這些對佈局最佳化都帶來了新而嚴酷的挑戰。延遲與雜訊的最佳化往往決定一個設計的成功與否，而增加導線尺寸與加大導線間距普遍的被應用來解決這兩個最佳化的問題。在設計流程的後端裡，經常須要執行後端設計變更(ECO)繞線來作延遲和雜訊的最佳化。而先前設計裡已存在龐大數目的障礙物與為因應不同雜訊與延遲問題而衍生各式各樣的設計規則使得 ECO 繞線變得非常困難。與網格式繞線器相較，非網格式繞線器更能夠克服前述問題的瓶頸。為了能夠快速執行數百萬閘級設計與系統單晶片的 ECO 繞線，論文中我們建置了一個有效率的方塊式點對點繞線器。

雖然方塊式繞線器在非點格式繞線器中擁有較簡單的繞線圖形，但是隨著晶片複雜度的增加，其繞線圖形複雜度也增加至百萬以上的節點數目，因此有效簡化在繞線時所處理的繞線圖形節點複雜度可大幅提高執行效率。在本論文中，我們提出兩種方法來加快方塊式繞線器的執行效率。第一種是繞線圖形節點的簡化。我們提出兩種方法不但可以減少方塊碎裂的情形，進而提高了繞線的執行效率而且不會因簡化繞線圖形而犧牲繞線品質。第二種是 ECO 全域繞線。我們提出不同的繞線資源評估方式以適用 ECO 繞線問題的特質，此外也提出由擴展繞線與全域細胞內部導線路徑重組與全域細胞重新規劃的繞線流程，除了保證一定可以找到存在的繞線路徑外，也由於限制了繞線搜尋的範圍，更可大大減少所需的繞線時間，不過會付出降低一些繞線品質的代價。實驗結果顯示，簡化繞線圖形可減少繞線時間約 40%;而進一步應用 ECO 全域繞線更可大幅減少繞線時間至 85%左右。

# An Efficient Tile-Based ECO Router for SoC Designs

Student: Jian-Yin Li

Advisor : Dr. Yih-Lang Li

Department of Computer and Information Science  
National Chiao Tung University

## ABSTRACT

Remarkable advances in the process and circuit designs bring crucial challenges for optimizing the layout of a multi-million gate design. Moreover, introducing System On a Chip (SOC) design methodology greatly increases the design complexity and the layout integration complexity of various Intelligent Properties (IPs). Delay and noise optimization are dominant factors to succeed in the design, where wire sizing and spacing are widely used for solving the problems respectively. Engineering Change Order (ECO) routing is frequently requested in the later design stage for the purpose of delay and noise optimization. ECO routing is very difficult as a result of huge existing obstacles and the requests for various design rules. Gridless routers are more applicable to overcome the problem barrier than grid routers. Therefore, we develop an efficient tile-based point-to-point router for the ECO routing of multi-million gate designs in this thesis.

Although tile-based routers have less number of nodes of routing graph than grid routers and connection-based routers, as the design complexity increases, the number of nodes of tile-based routing graphs have grown over thousand millions for SOC designs. We can reduce the complexity of tile-based routing graph to promote routing performance. In this thesis, we propose two methods to promote routing speed of the tile-based router. The first is *Routing Graph Reduction*. We propose two methods, i.e., redundant tiles removal and neighbor tiles alignment, to reduce the complexity of tile-based routing graph. It diminishes tile fragmentation as well as reduces the routing time without sacrificing routing quality. The second is *ECO Global Routing*. We propose different resource estimation scheme from that used by general global routing to reflect the characteristics of ECO routing problem. Also, we propose an ECO routing flow, including extended routing and GCell restructuring and rescheduling, to guarantee to find a feasible solution if there exists such a solution. By

limiting the searching scope of ECO routing, ECO global routing improves a lot the routing speed at little expense of routing quality. Experimental results show that routing graph reduction can decrease the routing time by 40% and ECO global routing with routing graph reduction can further decrease the routing time up to 85% or so.



# Acknowledgements

I am deeply grateful to my advisor, Dr. Yih-Lang Li for his continuous guidance, support, and ardent discussion throughout this research. His valuable suggestions help me to complete the thesis. Also I express my sincere appreciation to all classmates in my laboratory for their encouragement and help.

This thesis is dedicated to my parents and my families for their patience, love, encouragement, and long expectation.



# Contents

Abstract (in Chinese).....	I
Abstract (in English).....	II
Acknowledgements.....	IV
List of Figures .....	VI
List of Tables.....	VII
<b>1 Introduction .....</b>	<b>1</b>
1.1 ECO Routing.....	1
1.2 Gridless router.....	2
1.3 Our approach.....	6
<b>2 Tile-Based Router.....</b>	<b>8</b>
<b>3 Routing Graph Reduction (RGR).....</b>	<b>12</b>
3.1 Redundant Tiles Removal.....	12
3.2 Neighbor Tiles Alignment.....	18
<b>4 ECO Global Routing.....</b>	<b>29</b>
4.1 Global Routing Graph.....	30
4.2 Extended Routing and GCell Restructuring and Rescheduling.....	36
4.2.1 Extended Routing.....	38
4.2.2 GCell Restructuring and Rescheduling.....	41
<b>5 Experimental Results.....</b>	<b>45</b>
<b>6 Conclusions.....</b>	<b>48</b>
<b>Bibliography.....</b>	<b>49</b>

# List of Figures

1.1	Connection graph.....	3
1.2	Tile-based graph.....	5
1.3	ECO routing flows.....	6
2.1	Tile plane examples.....	9
2.2	Tile propagation.....	11
3.1	Redundant tiles.....	14
3.2	Enumeration order.....	14
3.3	Remove the redundant space tiles.....	16
3.4	Result of Redundant Tiles Removal.....	16
3.5	Remove the redundant space tiles.....	17
3.6	Neighbor Tiles Alignment.....	19
3.7	Result of Neighbor Tiles Alignment.....	19
3.8	Illegal alignment.....	21
3.9	Four shrinking cases.....	23
3.10	Shrinking Case (1).....	24
3.11	Shrinking Case (2).....	25
3.12	Shrinking Case (3).....	26
3.13	Shrinking Case (4).....	27
3.14	Final Result of Routing Graph Reduction.....	28
4.1	GCell and Global Routing Graph.....	31
4.2	Internal edges.....	32
4.3	(a) A path across a GCell without obstacles.	
	(b) A path across a GCell with existing obstacles.....	35
4.4	Tile propagation in the active GCell .....	35
4.5	Tile propagation status.....	37
4.6	GCell's location.....	39
4.7	Extended Routing.....	40
4.8	GCell Restructuring.....	42
4.9	GCell ReScheduling.....	43



# List of Tables

5.1	The number of tiles in corner stitching planes.....	44
5.2	The pre-process time before routing.....	44
5.3	Apply Routing Graph Reduction.....	46
5.4	Apply Routing Graph Reduction and ECO Global Routing.....	46

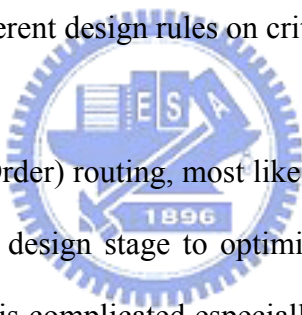


# Chapter 1

## Introduction

### 1.1 ECO Routing

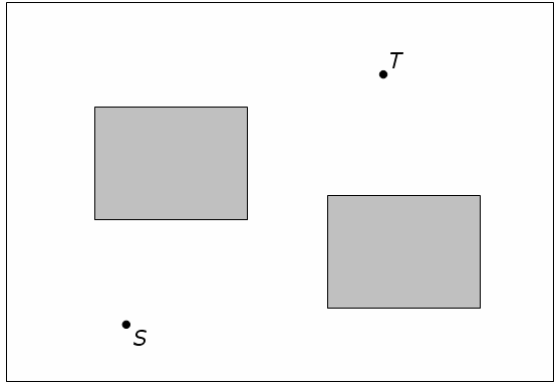
In the era of deep submicron (DSM) technology and SoC design, multi--million gate designs bring new challenges for layout optimization, where interconnect optimization becomes a dominant factor with the trend of ongoing shrinking in the device size, wire width and wire space. To solve the excesses of delay and noise, wire sizing and wire spacing have been surveyed [1]. It needs different design rules on critical or sensitive nets.



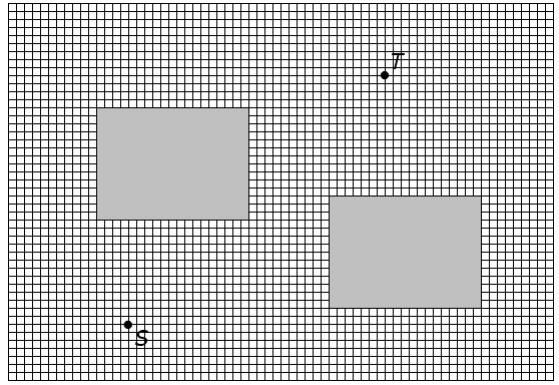
ECO (Engineering Change Order) routing, most likely a point-to-point routing operation, is a common request in the later design stage to optimize delay and noise or to complete an imperfect layout. ECO routing is complicated especially at top level of an SoC design. Three facts result in this predicament. The first is that ECO routing has to face a lot of interconnections produced by P&R (Place and Route) tools. Besides, ECO routing is unable to utilize the hierarchical structure, which can enormously diminish the preprocessing time, i.e., flattened traverse is necessary for ECO routing. The last is that ECO routing is unable to accede to the original routing environment, which forces ECO routing to extract its own database from the beginning. Only the designs migrated from different technology using existing mask layout or importing a hard IP will suffer from the incompatibility, however, layout migration and IP reuse have been common for current designs.

## 1.2 Gridless Router

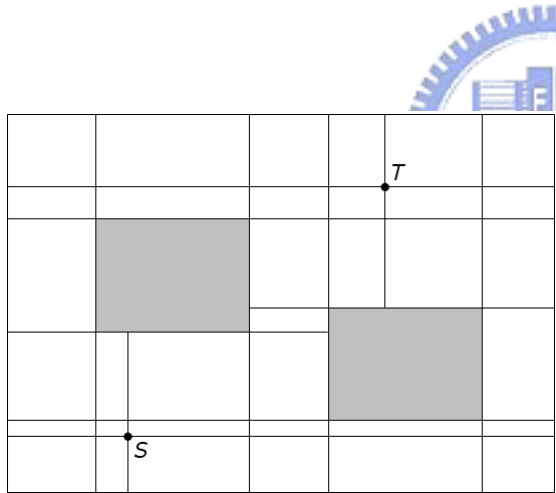
For a design with several hard IPs created under different routing environments, it has been beyond the power of grid router. Moreover, applying different design rules for separate ECO routings requires more routing flexibility. Gridless router undoubtedly conforms to these requirements. A straightforward realization of gridless router is using fine uniform grids, i.e., manufacturing grids. Figure 1-1(a) shows a layout with two obstacles, and S and T are routing terminals. Figure 1-1(b) shows the approach by fine grids. Although it can accommodate various routing rules, the induced huge graph of a big design is infeasible because it costs lots of memory and searching time. To reduce the routing graph, gridless routers have been well studied and developed [2]-[15] where connection graph and tile graph are the most popular models. Zheng[5] constructed connection graph by extending lines through the boundaries of all obstacles until intersecting with other obstacles or boundaries of routing region, as shown in Figure 1-1(c). But it is expensive to pre-construction and representation. Cong[5, 6, 7] presented a connection graph implicitly which extending lines may pass through obstacles, as shown in Figure 1-1(d). Cong's graph is built efficiently and guarantees find an optimal path although it has more nodes and edges than Zheng's. But it needs additional query operation for the legality of next move, and when the design has many layers and has complicated layout, the number of nodes of the graph is still much.



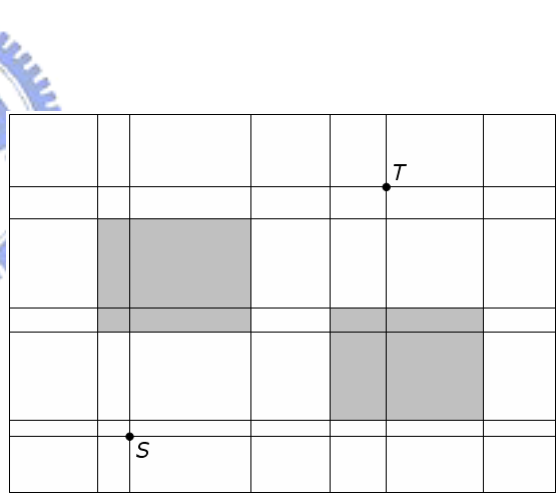
(a)



(b)



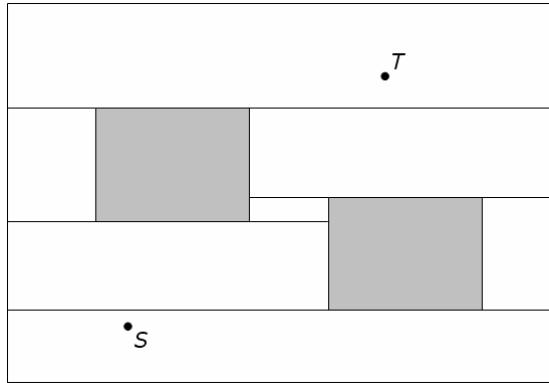
(b)



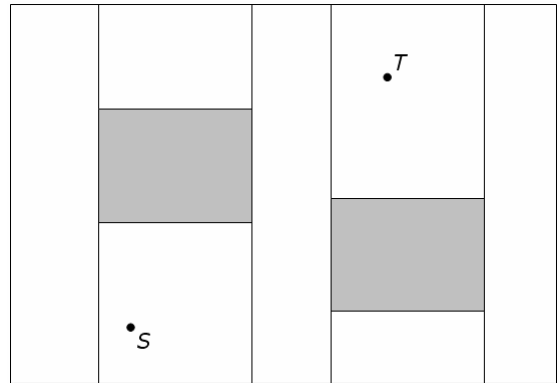
(d)

**Figure 1-1. Connection graph**

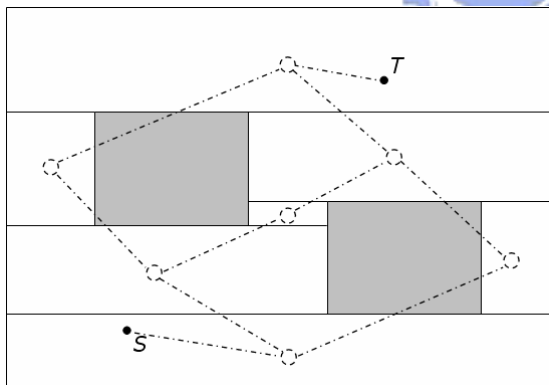
Tile-based graph [9]-[15] is another approach, where the routing region and obstacles are divided into space tiles and block tile respectively with corner-stitching data structure[16]. Figure 1-2 (a) shows the horizontal tile plane where each tile is the maximum horizontal strip and Figure 1-2 (b) shows the vertical tile plane where each tile is the maximum vertical strip. A space tile corresponds to a node of the routing graph and there exists an edge between two nodes if the related space tiles are adjacent, as shown in Figure 1-2 (c) and (d). Searching in tile graph is usually faster because the number of nodes of the routing graph is much lesser than Zheng and Cong's graph and each tile has four corner pointers to stitch its four corner neighbors such that query operations are very efficient. Main query operations in maze routing are neighbor finding and area enumeration whose complexities are  $O(\sqrt{n})$  where  $n$  is the number of tiles. Actually their average complexities are much lesser than  $\sqrt{n}$  because the operations are used locality since maze routing is to propagate node by node. The major drawback of tile-based approach is that it takes more time to build corner-stitching tile planes for all routing layers compared with connection graph approach. It can be improved by computational geometry method [17]. In [15], Xing applied a piecewise linear cost model to guide the maze operation and guaranteed to find an optimal path.



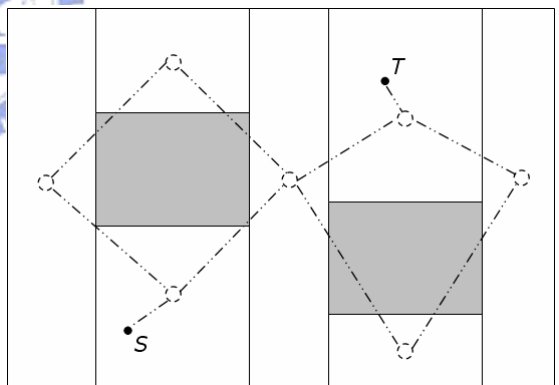
(a) Horizontal tile plane



(b) Vertical tile plane



(c) Corresponding routing graph



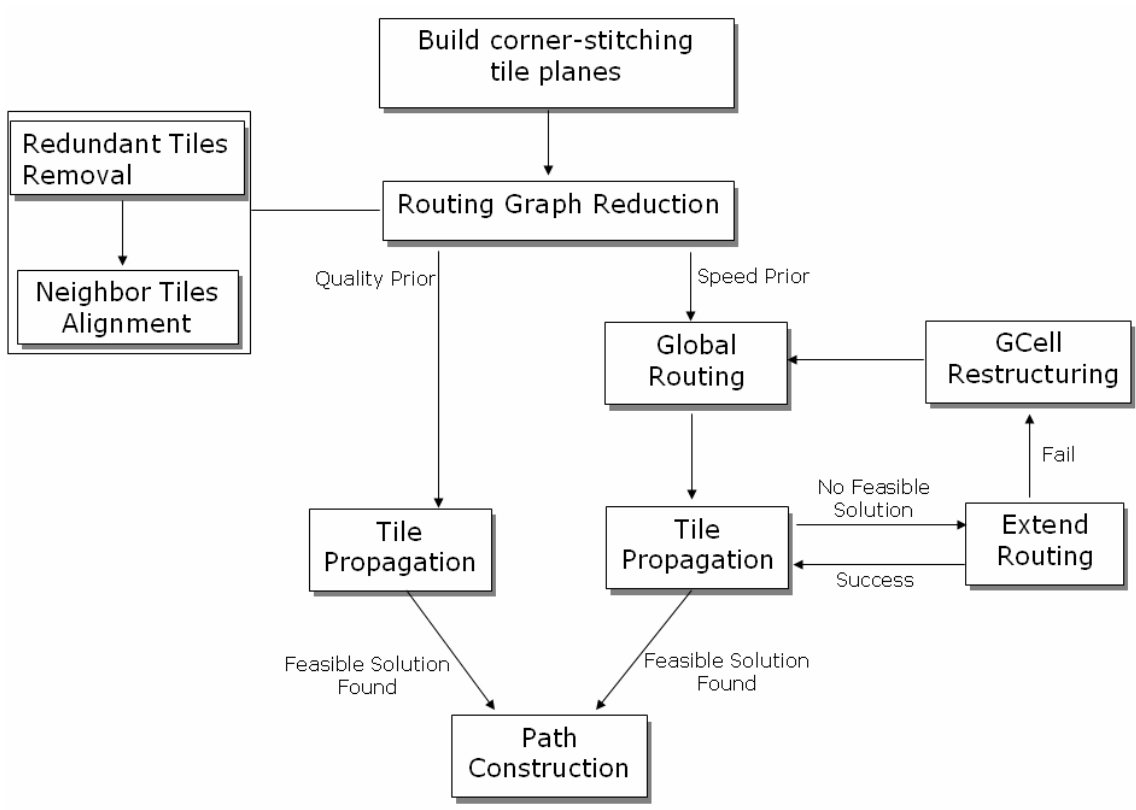
(d) Corresponding routing graph

**Figure 1-2. Tile-based graph**

### 1.3 Our Approach

Both connection graph and tile-based approaches can find an optimal path for an ECO point-to-point routing, but considerable runtime for a big design is not practical. For example, a chip-set design can produce several hundred millions of tiles for one routing layer. A fast and near optimal path of an ECO routing is much more practical for multi-million gate designs.

In this thesis, we focus on the ability of routing multi-million gate designs. We apply tile-based approach in our research due to the smaller number of nodes in the routing graph and efficient corner-stitching data structure. We propose two methods to promote routing speed of the tile-based router. The first is *Routing Graph Reduction (RGR)*. It diminishes tile fragmentation as well as reduces the routing time without sacrificing routing quality. The second is *ECO Global Routing*. It improves a lot the speed of the routing at little expense of routing quality. Figure 1-3 shows our ECO routing flow. We will explain each stage in the following Chapters. Chapter 2 presents the review of the tile-based router. The Routing Graph Reduction and ECO Global Routing are proposed in Chapter 3 and Chapter 4. Experimental results are reported in Chapter 5. Finally, we give conclusion in Chapter 6.



**Figure 1-3. ECO routing flows**

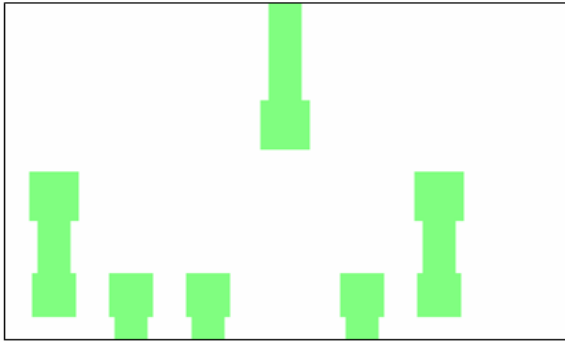


# Chapter 2

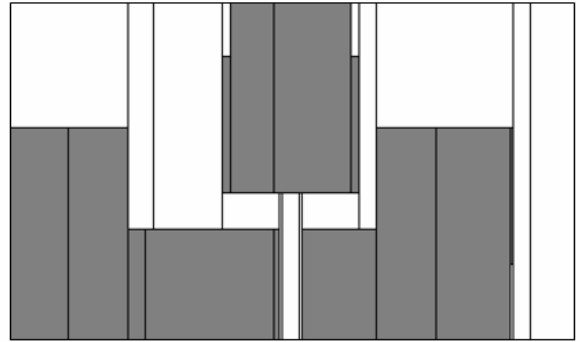
## Tile-Based Router

The tile-based routers have been well studied and developed to find a tile-to tile path in tile graph [9]-[15]. We apply Dion's approach [13] to establish our tile-based router and give a brief review in this chapter.

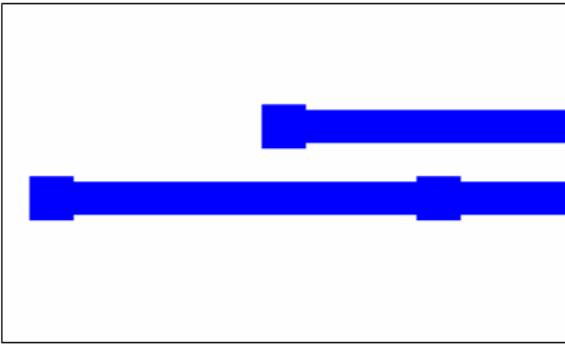
First we build corner-stitching tile plane for each routing layer. A tile plane can be built in maximum horizontal or vertical strip. The preferable routing direction of a metal layer is the same as the tile-stripped direction of that layer. The obstacles in tile plane are the over-sized raw objects by  $(w_s + w_w/2 - 1/2)$ , where  $w_s$  is the wire space of current net to all objects of the same layer and  $w_w$  is the wire width. It guarantees the centerline of a wire can pass through any space tile without violating design rules. For multi-layers routing, two tiles on adjacent layers can be connected if the legal via region exists inside their intersection area. The legal via region is extracted in a similar concept to the wire plane, i.e., the via region is the region where the center point of a via can be placed. Via size, via enclosing rules for adjacent metal layers are the sizing factors of via region extraction. Figure 2 shows a tile plane example. Figure 2 (b) shows the Metal2 tile plane and Figure 2 (d) shows the Metal3 tile plane. Figure 2 (f) shows the Via2 tile plane.



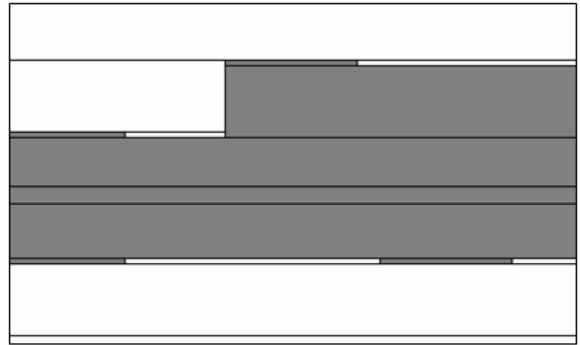
(a) Metal2 layer



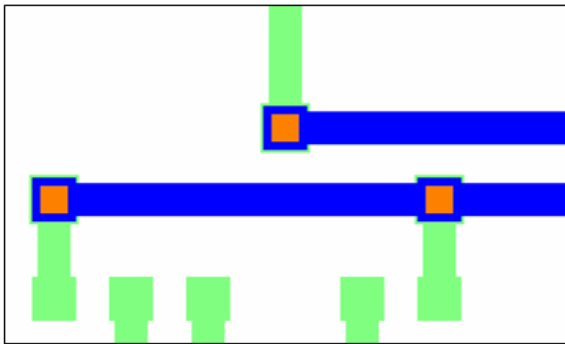
(b) Metal2 tile plane



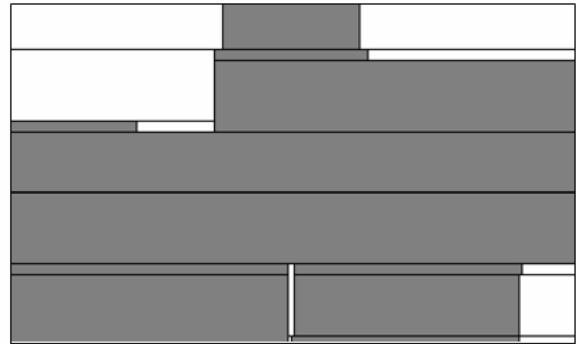
(c) Metal3 layer



(d) Metal3 tile plane



(e) Metal2 & Metal3 layer



(f) Via2 tile plane

**Figure 2-1. Tile plane examples**

Point-to-Point router consists of two stages: tile propagation and path construction. A tile can propagate to its neighbor space tile of the same layer or adjacent layer if there is a via region connecting these two space tiles. Tile propagation applies pre-defined cost function to guide the path searching, and then finds an optimal list of free tiles. For multi-layer routing, the tile list may contain tiles of different layers. Path construction constructs a minimum-corner path passing through the tile list.

Tile propagation records the path node whenever entering a new tile. The path node is the equipotential minimum cost segment from source. The path node of a tile is linked to the path node of its previous tile by a backward pointer. The path node is used to estimate the path cost and the cost to target. The start point can be a point, an edge, or a shape. The router will find a minimum cost path to connect two shapes if selected objects are shapes. Figure 2-2 shows an example of tile propagation. The source terminal tile  $S$  propagates into its three neighbor space tiles, say  $C_1$ ,  $C_2$ , and  $C_3$ , by their relative path nodes  $P_1$ ,  $P_2$ , and  $P_3$  that are pushed into a priority heap, say  $H_p$ . The path node with minimum cost is then popped from  $H_p$  after  $S$  completes its propagation. This process repeats until target terminal tile  $T$  is visited and the path cost is lesser than other path nodes in  $H_p$ . Therefore, we can find the optimal tile list by the backward pointer. If  $H_p$  becomes empty before  $T$  is visited, there is no feasible path from  $S$  to  $T$ .

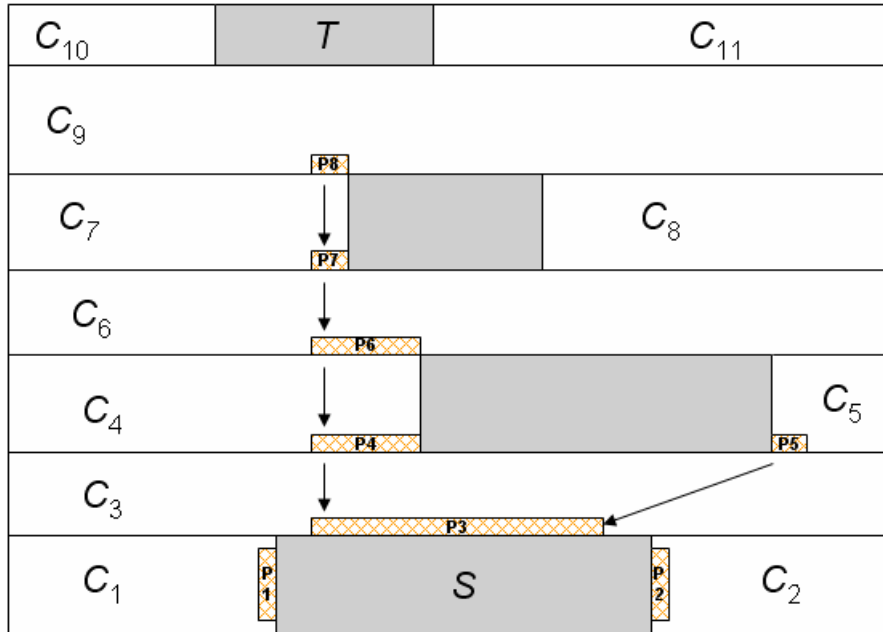



Figure 2-2. Tile propagation

# Chapter 3

## Routing Graph Reduction (RGR)

Tile propagation, the way of tile-based router to find a routing path, explores to reach the target all possible ways over the tiles of the same layer and across neighboring layers. Generally, the tile planes of a design with many existing obstacles are fragmented. This phenomenon increases the computation complexity of tile propagation. We propose two methods, i.e., redundant tiles removal and neighbor tiles alignment, to diminish tile fragmentation as well as to reduce the tile-propagation time in this thesis.

### 3.1 Redundant Tiles Removal



By observing the tile propagation, we found that further propagations of many paths are terminated because those paths enter a space tile that has no exit for further propagation. And most of the cases, the routing graph consists of many connected graphs rather than a connected graph. Those connected components not containing the terminals to be routed will not be visited during tile propagation. Such space tiles do not contribute to routing rather than increases tile fragmentation problem. The basic concept of redundant tiles removal is to remove these tiles.

We define some terminologies used in this thesis as follows:

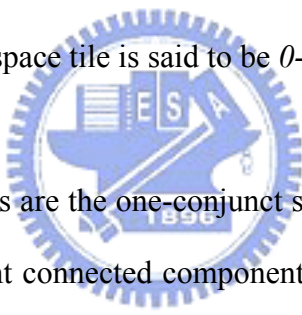
**Definition 3-1. (essential/redundant tile)** If a space tile can contribute to further propagation, we call it an *essential tile*, otherwise, we call it a *redundant tile*.

**Definition 3-2. (essential/redundant connected component)** We call the connected component of the routing graph containing the terminals to be routed an *essential connected component*, and the others *redundant connected components*.

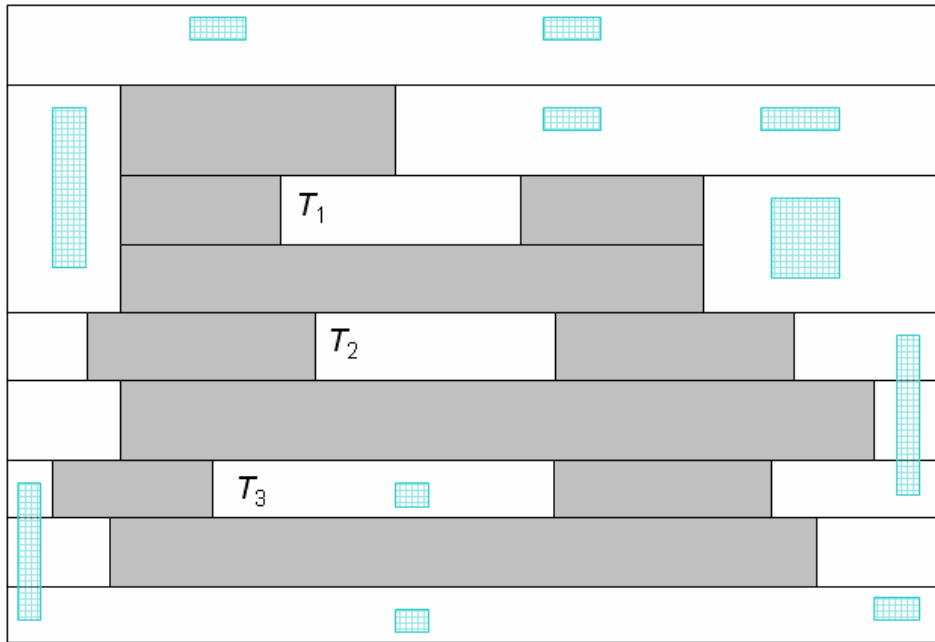
**Definition 3-3. (conjunct tile)** A tile,  $A$ , is referred to a *conjunct tile* of a tile,  $B$ , if tile propagation, from  $A$  to  $B$ , on the same layer or across adjacent layer is feasible.

**Definition 3-4. (one-conjunct)** A space tile is said to be *one-conjunct* if it has only one conjunct tile.

**Definition 3-5. (0-conjunct)** A space tile is said to be *0-conjunct* if it has no conjunct tile.



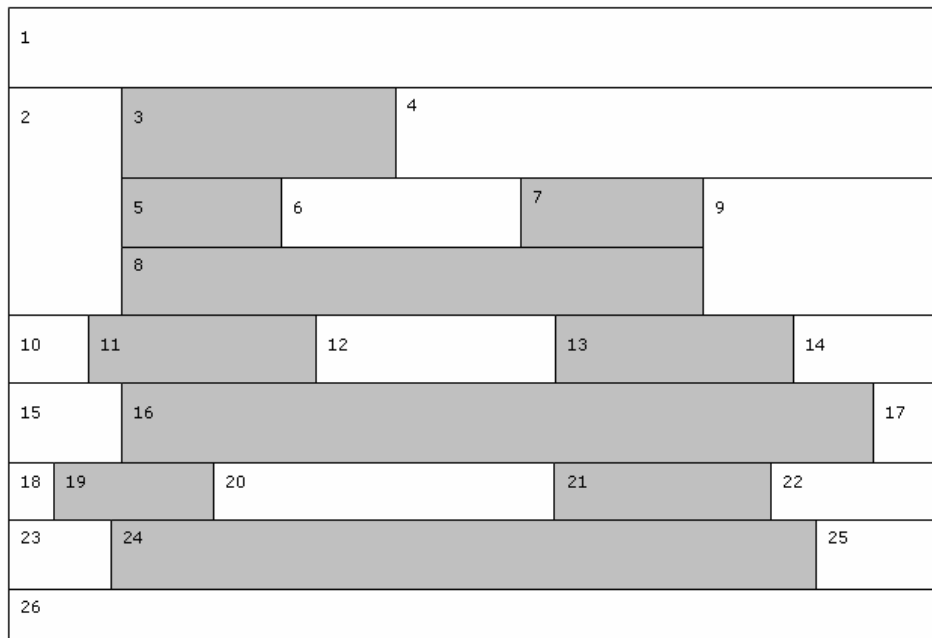
Obviously, the redundant tiles are the one-conjunct space tiles and the 0-conjunct space tile related to a node of a redundant connected component. We show redundant tiles,  $T_1$ ,  $T_2$ , and  $T_3$ , in Figure 3-1 that shows a simple tile plane of a metal layer. Tiles  $T_1$  and  $T_3$  are the one-conjunct space tiles and tile  $T_2$  is a 0-conjunct space tile. Note that  $T_3$  is accessible from only one tile of other layer through the via region. If  $T_3$  can be accessible from more than two tiles of other layer through the via region, it is an essential tile.



Block Tile
  Space Tile
  Via Region

Block Tile : 10    Space Tile: 16

**Figure 3-1. Redundant tiles**



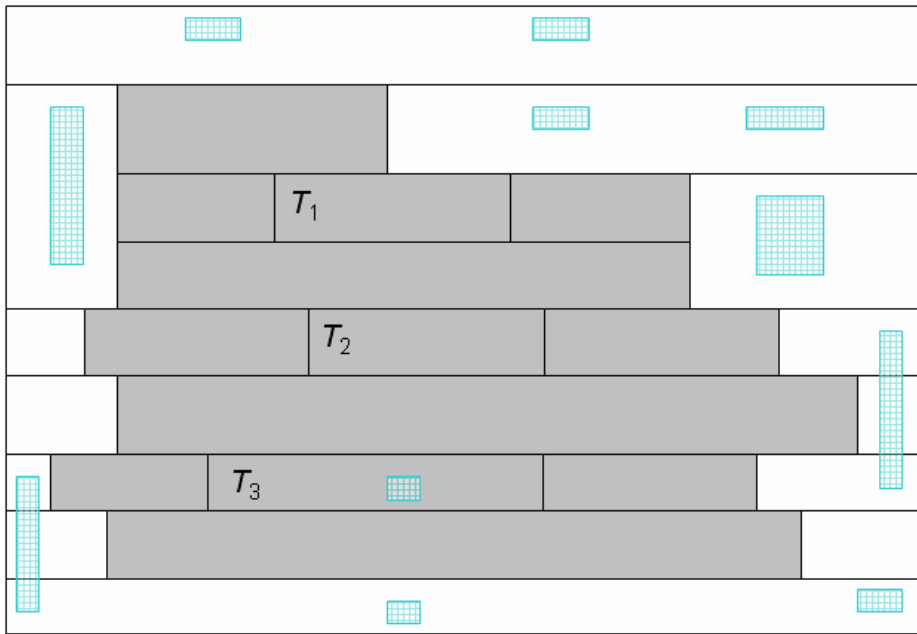
**Figure 3-2. Enumeration order**

The corner-stitching data structure provides fast neighbor finding and area enumeration operations, so we can efficiently check neighbor space tiles and via region of a space tile. Also we know that the enumeration operation visits each tile exactly once, as shown in Figure 3-2. Therefore, we can remove the redundant tiles by examining one-conjunct and 0-conjunct space tiles within an enumeration operation over the whole tile plane. It is worth to note that we only change the redundant space tiles to be block tiles without merging them with their neighbors to preserve maximum horizontal strip during enumeration, as shown in Figure 3-3. Merging during enumeration may disorder the enumerating order. After the completion of enumeration operation, we reconstruct the tile plane to preserve the maximum horizontal or vertical strip, as shown in Figure 3-4.

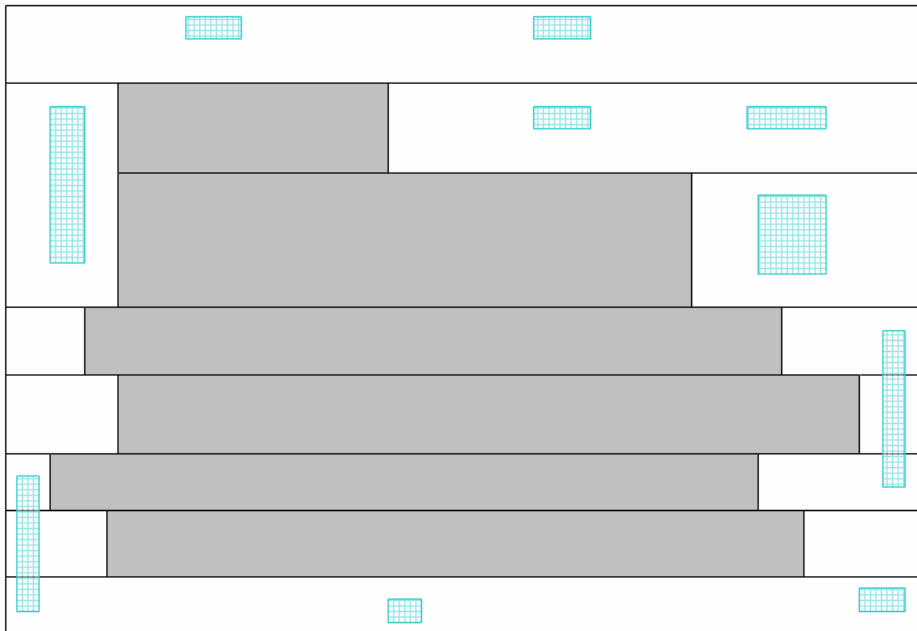
There is one situation worth of paying more attention to deal with during redundant tile removal process. Figure 3-5 (a) shows such a situation. Initially, tiles  $T_1$  and  $T_2$  are essential tiles. After removing redundant tile  $T_3$ ,  $T_2$  becomes a redundant tile. Also  $T_1$  becomes a redundant tile after removing  $T_2$ . This process is illustrated from Figure 3-5 (b) to Figure 3-5 (d).

Before applying redundant tile removal process, there are 10 block tiles and 16 space tiles in Figure 3-1. The redundant tile removal diminishes the tile fragmentation such that the final layout only contains 6 block tiles and 13 space tiles, as shown in Figure 3-4.



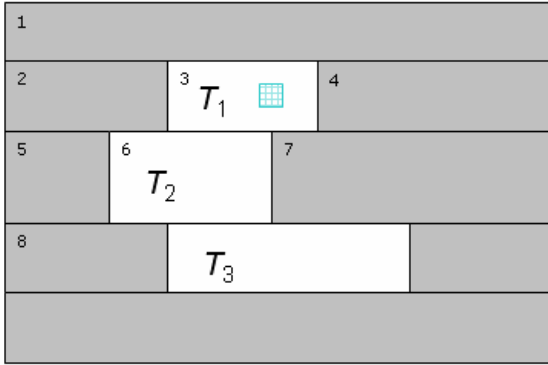


**Figure 3-3. Remove the redundant space tiles**

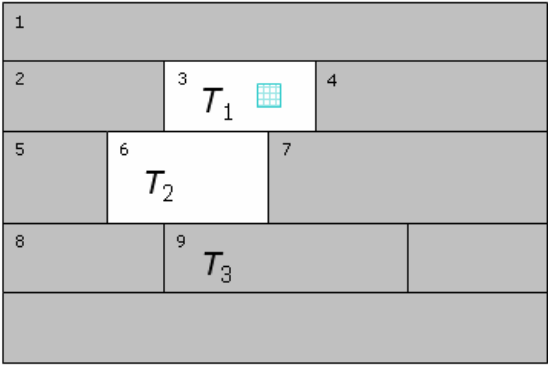


Block Tile : 6    Space Tile: 13

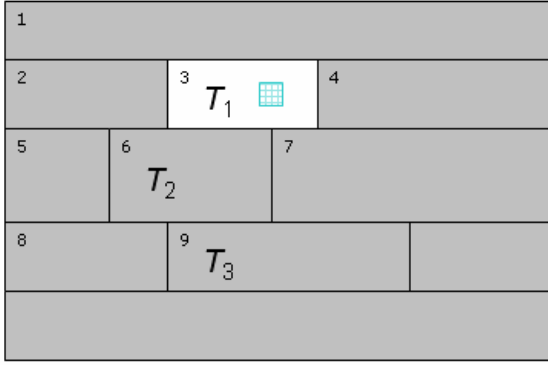
**Figure 3-4. Result of Redundant Tiles Removal**



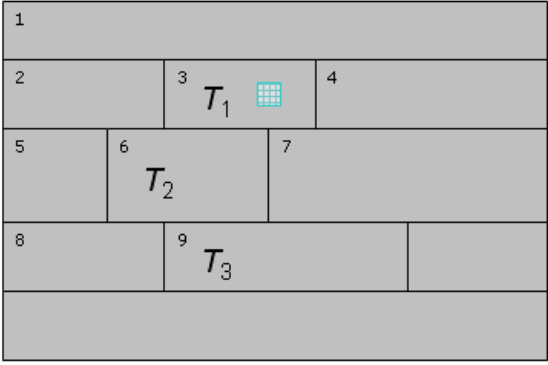
(a)



(b)



(c)



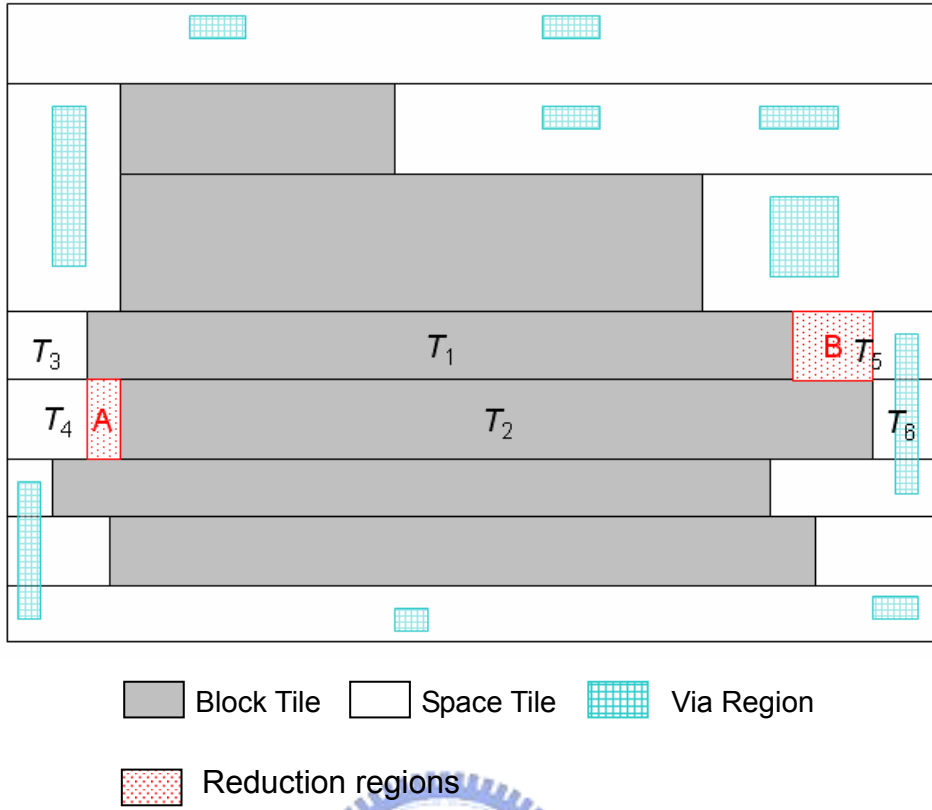
(d)

Figure 3-5. Remove the redundant space tiles

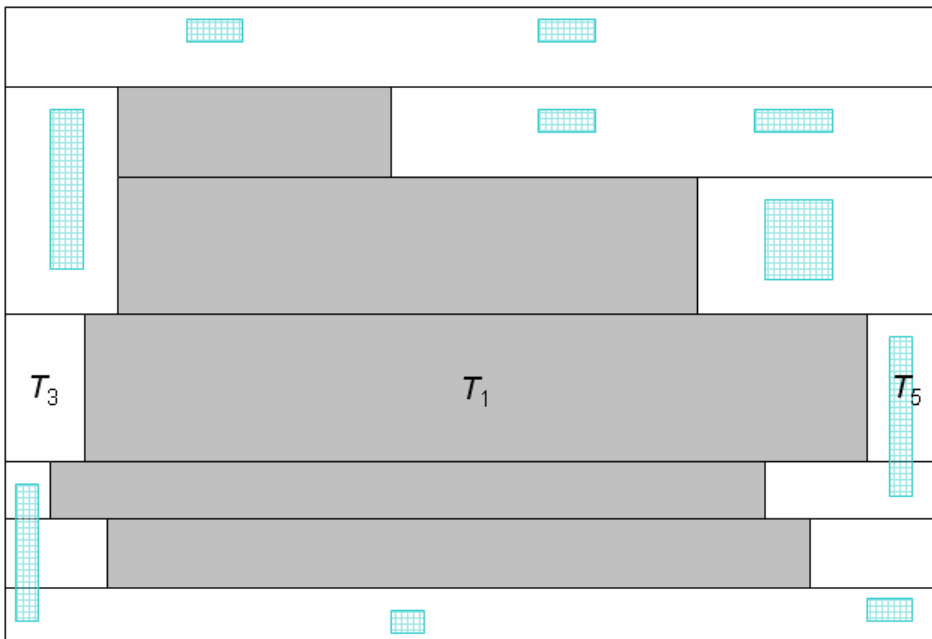
### 3.2 Neighbor Tiles Alignment

When we construct tile plane for a metal layer, the plane is sliced along the direction of metal layer's preferred routing direction so as to produce a maximum unidirectional strip. We illustrate the tile fragmentation problem using the maximum horizontal strip tile plane. Such a tile plane has the following characteristics: the top and bottom boundaries of the union area of those tiles that belong to a connected component on the same layer are flat, however, its left and right boundaries are often ragged. Ragged boundaries induce tile fragmentation. The basic idea is to shrink space tiles to flat the ragged border such that their neighbor block tiles can merge.

Figure 3-6 shows such an example. Tiles  $T_4$  and  $T_5$  are shrunk, and, meanwhile, tiles  $T_1$  and  $T_2$  are enlarged such that they can merge to a tile, where  $A$  and  $B$  are the reduction regions, a leftward shrinking is applied to  $T_4$ , and a rightward shrinking is applied to  $T_5$ . Figure 3-7 shows an advanced tile de-fragmentation if the left edges of tiles  $T_3$  and  $T_4$  have been aligned and the right edges of tiles  $T_5$  and  $T_6$  also can be aligned.



**Figure 3-6. Neighbor Tiles Alignment**



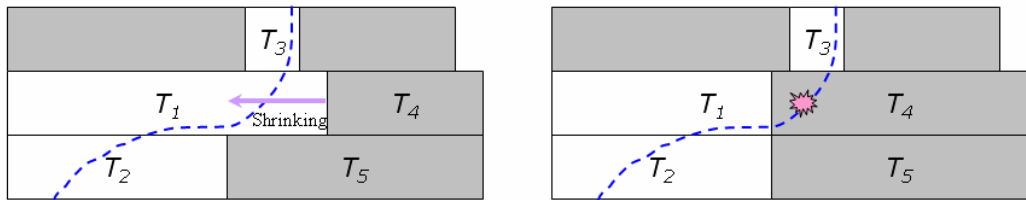
**Figure 3-7. Result of Neighbor Tiles Alignment**

Note that not all space tiles can be shrunk. If we want to shrink a space tile, say  $T_1$ , so as to merge with other tile, say  $T_2$ . Only the shrinking that preserves the following two properties is feasible.

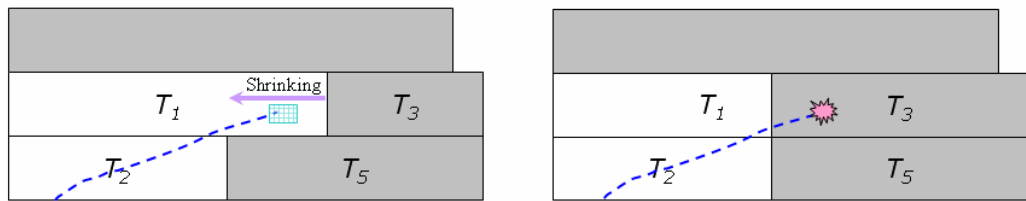
*Property 1.* Shrinking will not obstruct an existing path from  $T_1$  to its neighbor tile of the same layer, say  $T_3$ . Figure3-8(a) shows an illegal shrinking that does not preserve this property. In Figure 3-8(a), there is an original routing path between tiles  $T_1$  and  $T_3$ , however, this path will disappear if we perform a leftward shrinking on  $T_1$  to align with  $T_2$ .

*Property 2.* Shrinking will not obstruct an existing path from  $T_1$  to other tile of adjacent layer. This property requires there exists no via region overlapping with the reduction region. In Figure 3-8(b), a leftward shrinking on  $T_1$  will wipe out a routing path connecting to adjacent layer.





(a)



(b)

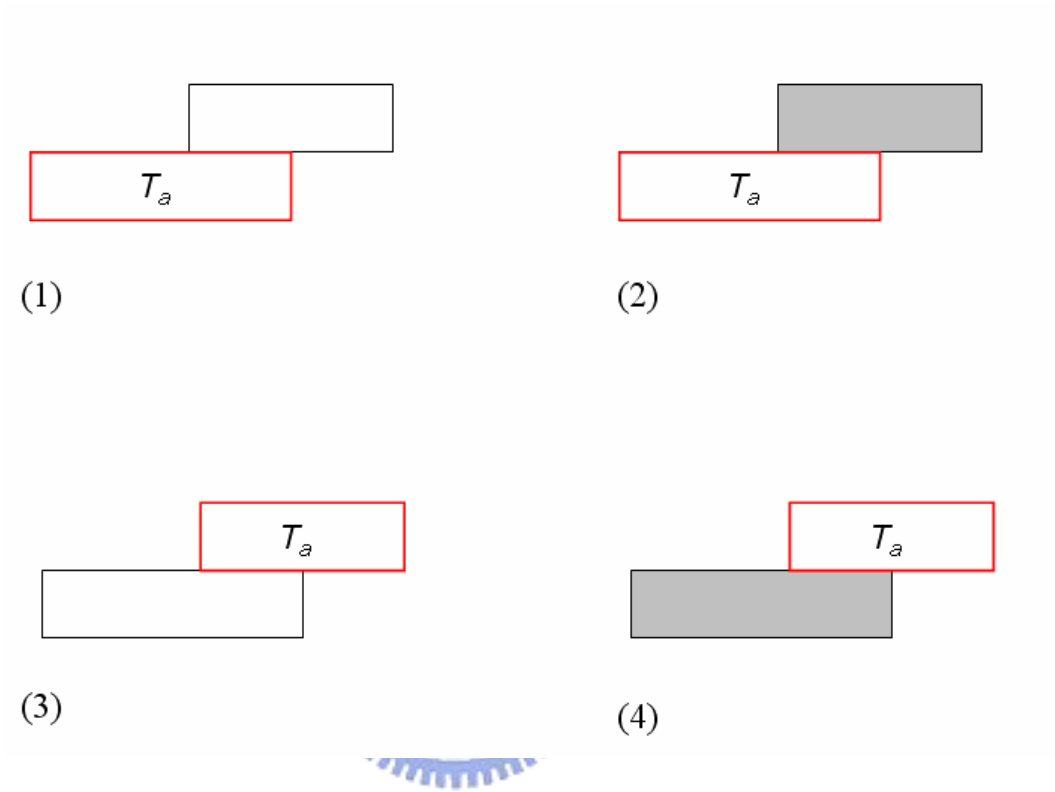
Block Tile
  Space Tile
  Via Region

**Figure 3-8. Illegal alignment**

Before introducing shrinking process, we first define some terminologies as follows.

- $l(T_i)/r(T_i)$ : the  $x$ -coordinate value of tile  $T_i$ 's left/right edge.
- $R_{LS}(T_i)/R_{RS}(T_i)$ : the leftward/rightward shrinking on  $T_i$ .
- $l(R_{LS}(T_i))/r(R_{LS}(T_i))$ : the stop/start position of a leftward shrinking on  $T_i$ .
- $l(R_{RS}(T_i))/r(R_{RS}(T_i))$ : the start/stop position of a rightward shrinking on  $T_i$ .
- $N_{rt}(T_i)$ : the rightmost top neighbor of tile  $T_i$ .
- $N_{tr}(T_i)$ : the topmost right neighbor of tile  $T_i$ .
- $N_{lb}(T_i)$ : the leftmost bottom neighbor of tile  $T_i$ .
- $N_{bl}(T_i)$ : the bottommost left neighbor of tile  $T_i$ .

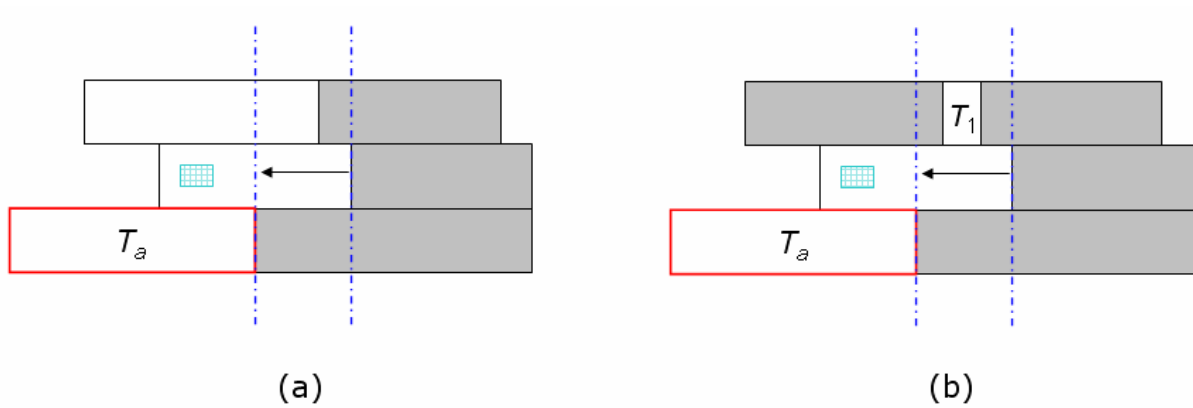
We list in Figure 3-9 four shrinking cases during enumeration to discuss how to decide if a shrinking on a space tile is feasible. We refer the *active tile*,  $T_a$ , as the space tile currently to be processed. The reason why we only process space tiles is that shrinking unnecessary space region will not reduce routability nor produce incorrect routing result. On the contrary, shrinking block tiles may produce a path across existing blockages. These cases are differentiated by  $T_a$ 's top and bottom neighbors.



**Figure 3-9. Four shrinking cases**

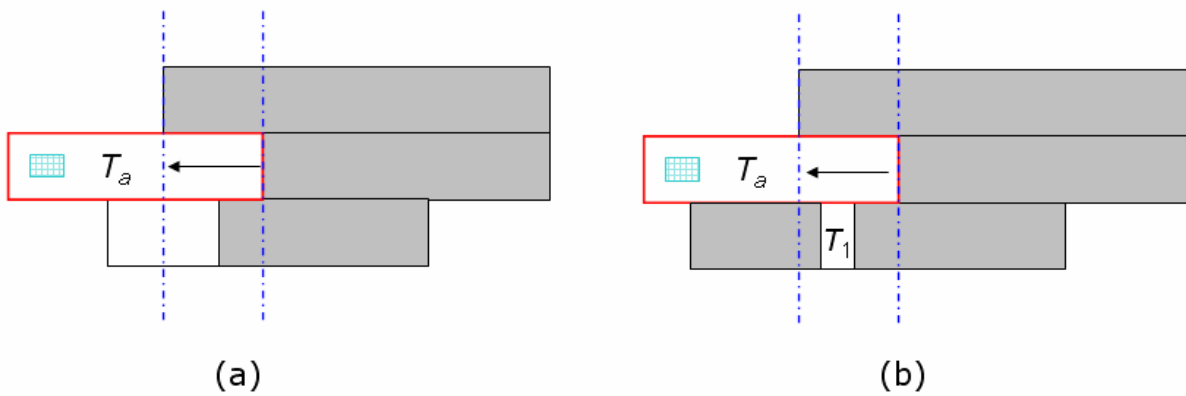


Case (1):  $N_{ri}(T_a)$  is a space tile and the candidate to be shrunk. The goal is to perform an  $R_{LS}(N_{ri}(T_a))$ , where  $l(R_{LS}(N_{ri}(T_a))) = r(T_a)$  and  $r(R_{LS}(N_{ri}(T_a))) = r(N_{ri}(T_a))$ . To preserve *Property 1*, there can not exist a top or bottom neighbor space tile of  $N_{ri}(T_a)$ , say  $T_1$ , such that  $l(R_{LS}(N_{ri}(T_a))) \leq l(T_1) < r(R_{LS}(N_{ri}(T_a)))$ . Figure 3-10(a) shows a legal shrinking, while Figure 3-10(b) shows an illegal shrinking.



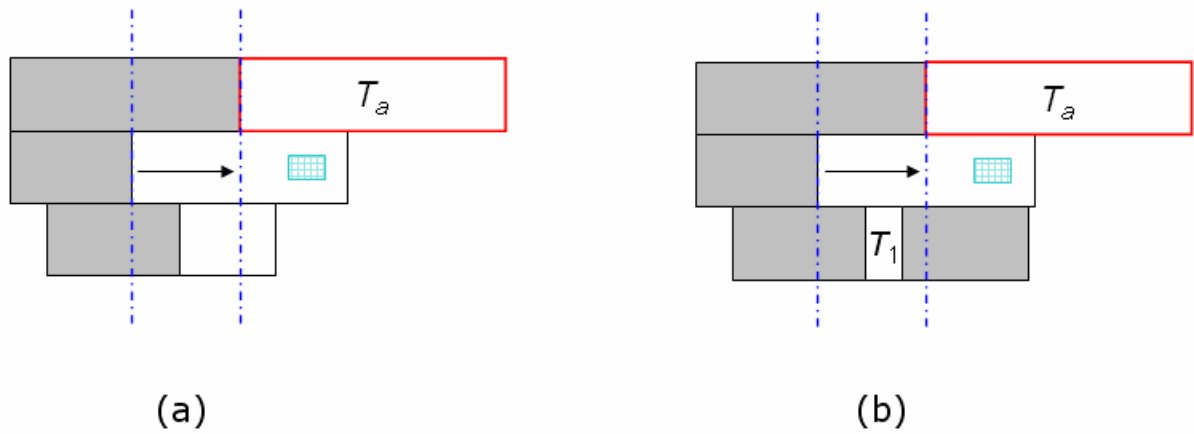
**Figure 3-10. Shrinking Case (1)**

Case (2):  $N_{rt}(T_a)$  is a block tile and  $T_a$  is the candidate to be shrunk. The goal is to perform an  $R_{LS}(T_a)$ , where  $l(R_{LS}(T_a)) = l(N_{rt}(T_a))$  and  $r(R_{LS}(T_a)) = r(T_a)$ . To preserve *Property 1*, there can not exist a bottom neighbor space tile of  $T_a$ , say  $T_1$ , such that  $l(R_{LS}(T_a)) \leq l(T_1) < r(R_{LS}(T_a))$ . Figure 3-11(a) shows a legal shrinking, while Figure 3-11(b) shows an illegal shrinking.



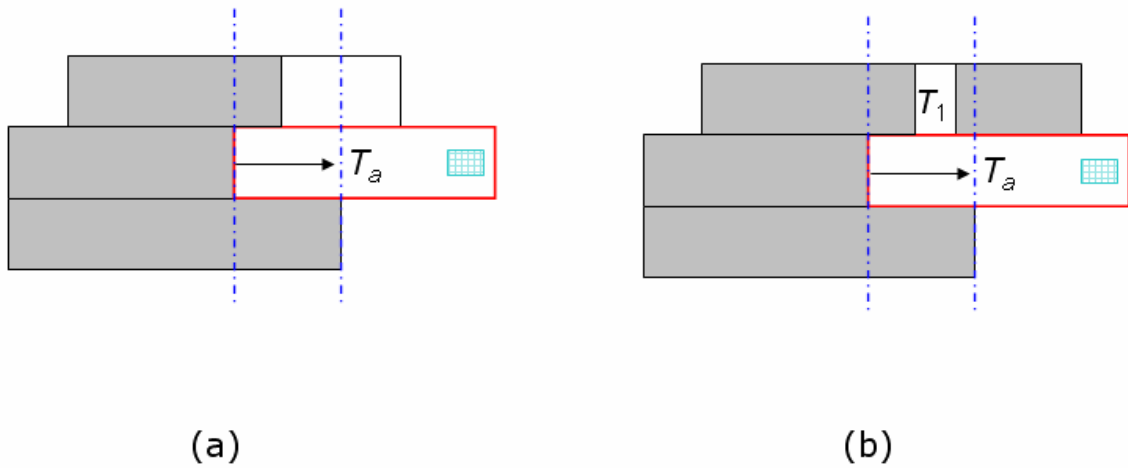
**Figure 3-11. Shrinking Case (2)**

Case (3).  $N_{lb}(T_a)$  is a space tile and the candidate to be shrunk. The goal is to perform an  $R_{RS}(N_{lb}(T_a))$ , where  $l(R_{RS}(N_{lb}(T_a))) = l(N_{lb}(T_a))$  and  $r(R_{RS}(N_{lb}(T_a))) = l(T_a)$ . To preserve *Property 1*, there can not exist a top or bottom neighbor space tile of  $N_{lb}(T_a)$ , say  $T_1$ , such that  $l(R_{RS}(N_{lb}(T_a))) < r(T_1) \leq r(R_{RS}(N_{lb}(T_a)))$ . Figure 3-12(a) shows a legal shrinking, while Figure 3-12(b) shows an illegal shrinking.



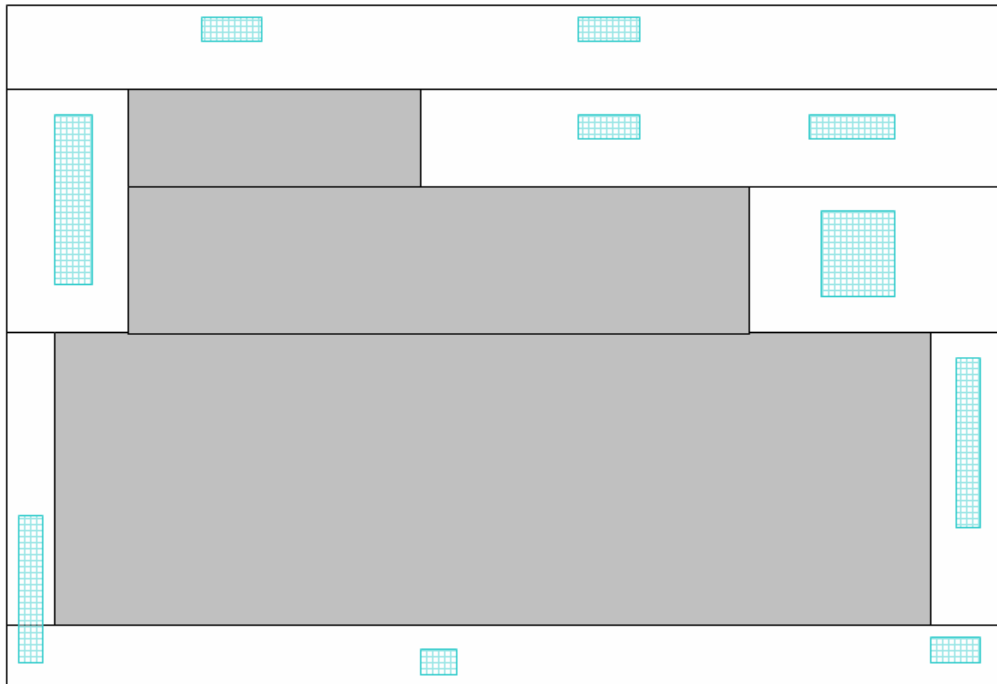
**Figure 3-12. Shrinking Case (3)**

Case (4).  $N_{lb}(T_a)$  is a block tile and  $T_a$  is the candidate to be shrunk. The goal is to perform an  $R_{RS}(T_a)$ , where  $l(R_{RS}(T_a)) = l(T_a)$  and  $r(R_{RS}(T_a)) = r(N_{lb}(T_a))$ . To preserve *Property 1*, there can not exist a top neighbor space tile of  $T_a$ , say  $T_1$ , such that  $l(R_{RS}(T_a)) < r(T_1) \leq r(R_{RS}(T_a))$ . Figure 3-13(a) shows a legal shrinking, while Figure 3-13(b) shows an illegal shrinking.



**Figure 3-13. Shrinking Case (4)**

In order to keep correct enumeration order, we just shrink tiles during enumeration. After finishing enumeration, we reconstruct the tile plane to preserve the maximum horizontal strip. Figure 3-14 shows the final result of the tile plane in Figure 3-1, whose total number of tiles is 26, after removing redundant tiles and aligning neighbor tiles. The numbers of block tiles and space tiles decrease to 3 and 7, respectively. Compared with the tile plane in Figure 3-1, the total number of tiles after applying routing graph reduction decreases by over 50%.



Block Tile : 3 Space Tile: 7

**Figure 3-14. Final Result of Routing Graph Reduction**

# Chapter 4

## ECO Global Routing

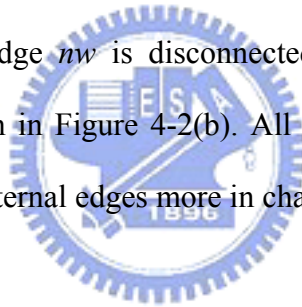
We present a method to diminish tile fragmentation in previous chapter. It reduces the tile propagation time without sacrificing routing quality. On the contrary, if the routing speed is a more important factor, we can improve a lot the speed of the routing at little expense of routing quality by introducing global routing concept. We apply global routing to reach this goal in this section. Global routing is used to guide the detail routing, and the routing resource on a layer is the major factor to estimate the routability of a routing region. However, it is quite different for an ECO routing because there are lots of existing obstacles. Therefore, the strategies used for an ECO global routing must also be different from those for block- or chip-level routing.

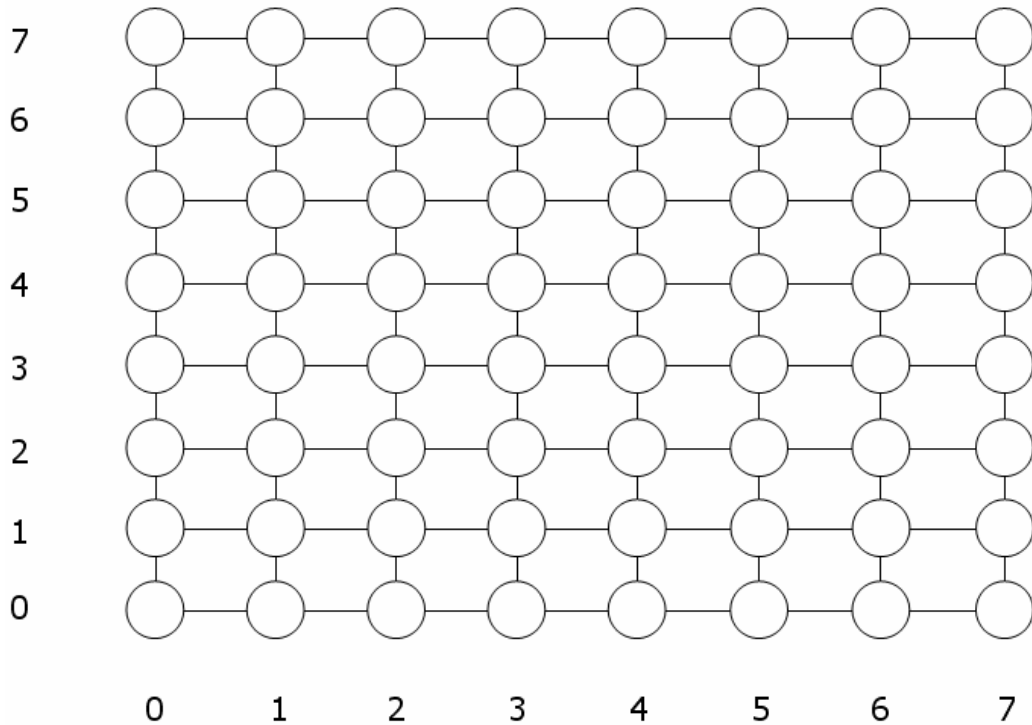
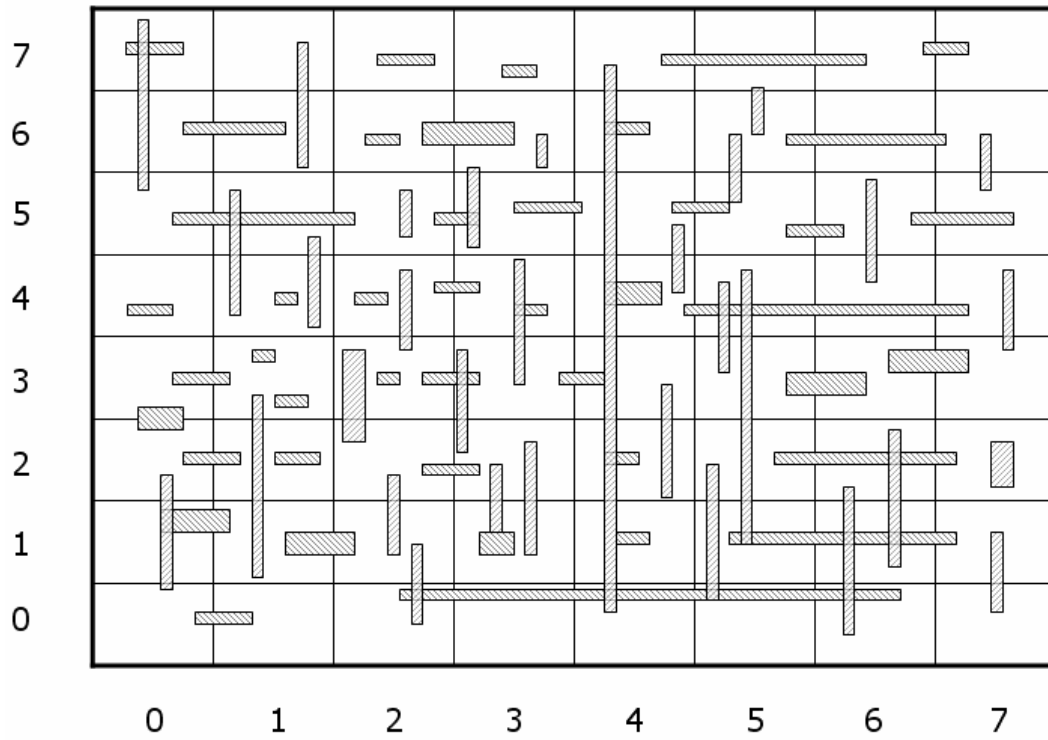


In Section 4.1, we introduce the global routing graph and the ECO global routing. In Section 4.2, we introduce two methods used to help ECO global routing finish the routing when initial global routing result can not guide detailed routing to find a feasible path.

## 4.1 Global Routing Graph

We partition the entire layout into several *global cells* (*GCell*) shown in Figure 4-1(a). The  $GC(i,j)$  is defined as the GCell at  $i$ -th column and  $j$ -th row. A global routing graph  $G(V,E)$  constructed by GCells is drawn in Figure 4-1(b). Each partition corresponds to a node and there exists an edge between two nodes if their related partitions abut. The  $V(i,j)$  is defined as the vertex at  $i$ -th column and  $j$ -th row. Besides, each vertex has six internal edges,  $nw$ ,  $ws$ ,  $se$ ,  $en$ ,  $ns$ , and  $ew$ , as shown in Figure 4-2(a). The  $nw$  edge represents the connectivity between the partition's north and west boundaries. The connectivity of an internal edge is either *connected* or *disconnected*. A connected  $nw$  edge of a partition, say  $P_i$ , stands for that there exists a routing path across  $P_i$ , that connects  $P_i$ 's north and west neighboring partitions. For example, if  $V(i,j)$ 's internal edge  $nw$  is disconnected, there is no path connecting nodes  $V(i,j+1)$  and  $V(i-1,j)$ , as shown in Figure 4-2(b). All internal edges are set to be connected initially. We will discuss the internal edges more in chapter 4.2.2.

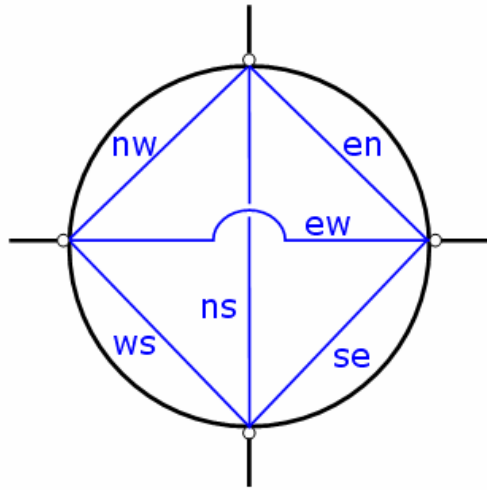




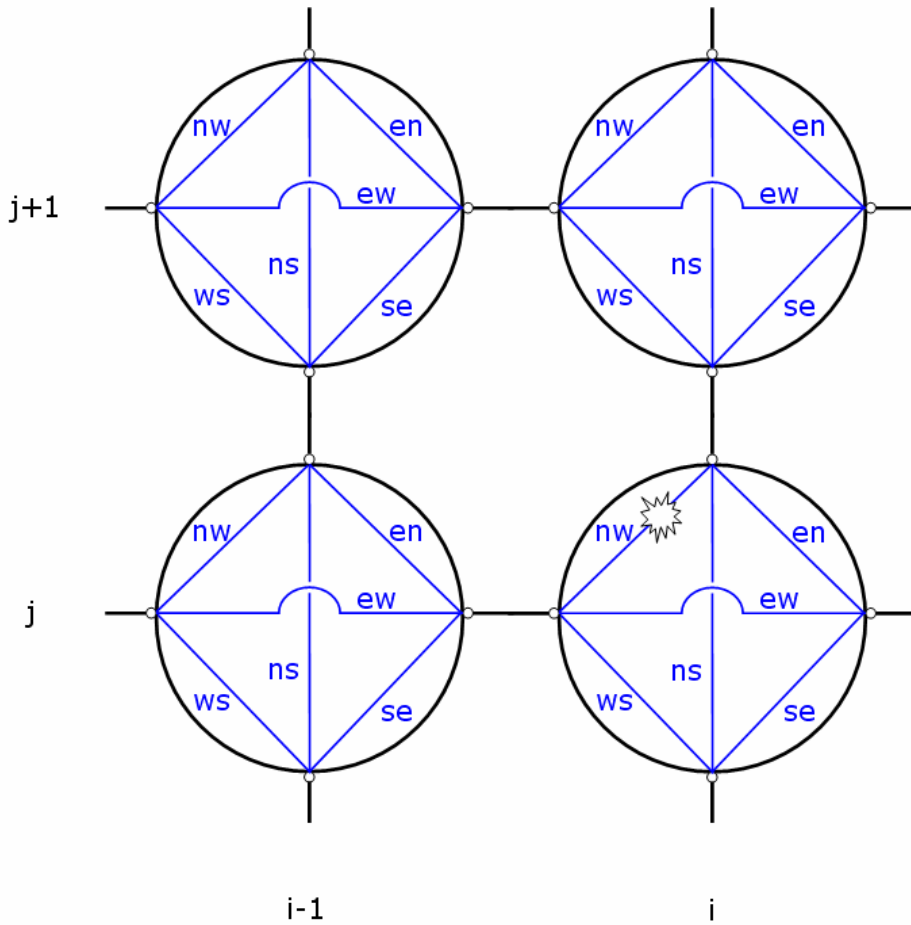
(b)

**Figure 4-1. GCell and Global Routing Graph**





(a)



(b)

Figure 4-2. Internal edges

Traditionally, each edge  $e_{ij}$  between vertices  $v_i$  and  $v_j$  has a capacity to represent the number of horizontal tracks or vertical tracks available between the corresponding GCells, but this does not provide enough information for ECO routing. A large amounts of existing obstacles fragment available routing regions, so it is often hard to find a straight path across a GCell and the path may be zigzag. Figure 4-3 shows this phenomenon. Via regions play an important role to find a path across a GCell. If a GCell has more via resources, it is easy to find a path passing through it. We estimate via resources by the total area of via regions that is a fast calculation by enumerating space tiles of the via tile planes. Via capacity,  $VC$ , of a GCell is defined as

$$VC = VA/A,$$

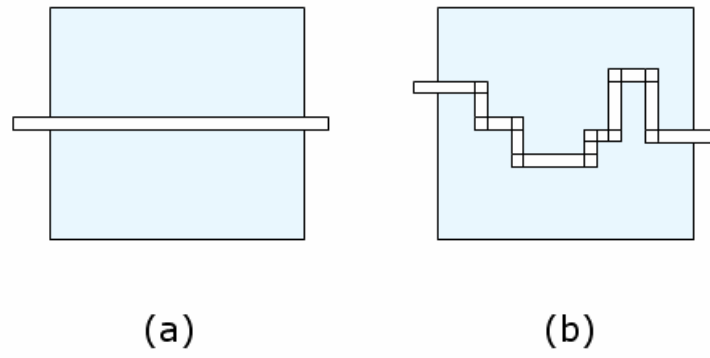
where  $VA$  is the total area of via region of the GCell, and  $A$  is the area of the GCell. We assign each vertex in G a cost,  $c$ , that is inverse proportion to  $VC$ . Cost  $c$  is defined by a piecewise linear function as

$$c = \begin{cases} 1/VC & \text{if } VC > t \\ k/VC & \text{if } VC \leq t \end{cases}$$

where  $t$  is a threshold, and  $k$  is an amplification scalar. By experimental observations, it is hard to get a path through the GCell when  $VC < 0.01$ , so we amplify the cost by multiplying a scalar. Also, we assign each edge of G a length cost,  $lc$ , to reflect the net timing factor.

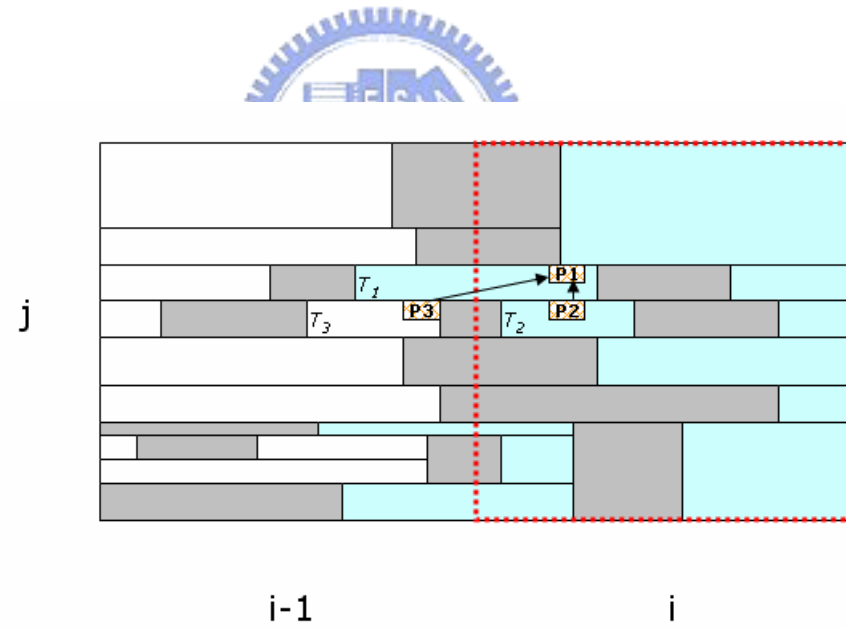
We apply the Dijkstra's algorithm in  $G$  to find the minimum-cost path connecting two terminals to be routed. We call the GCells on the minimum-cost path *active GCells*, and the other *idle GCells*. The nodes of  $G(V,E)$  are also referred to *active nodes* and *idle nodes* in the same way as their corresponding GCells. The further propagation out of active GCells is prohibited. To reserve the paths propagating into idle GCells, each idle GCell stores those paths entirely entering its region in its own *idle-path heap*. In Figure 4-4,  $GC(i,j)$  is an active GCell, while  $GC(i-1,j)$  is not. The path node  $p1$  can propagate to tiles  $T_2$  and  $T_3$  at  $p2$  and  $p3$ , respectively. Since  $p3$  entirely enters an idle GCell,  $p3$  is inserted into the idle-path heap of  $GC(i-1,j)$ .





**Figure 4-3. (a) A path across a GCell without obstacles.**

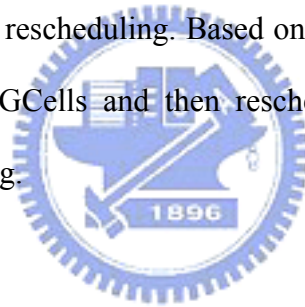
**(b) A path across a GCell with existing obstacles.**

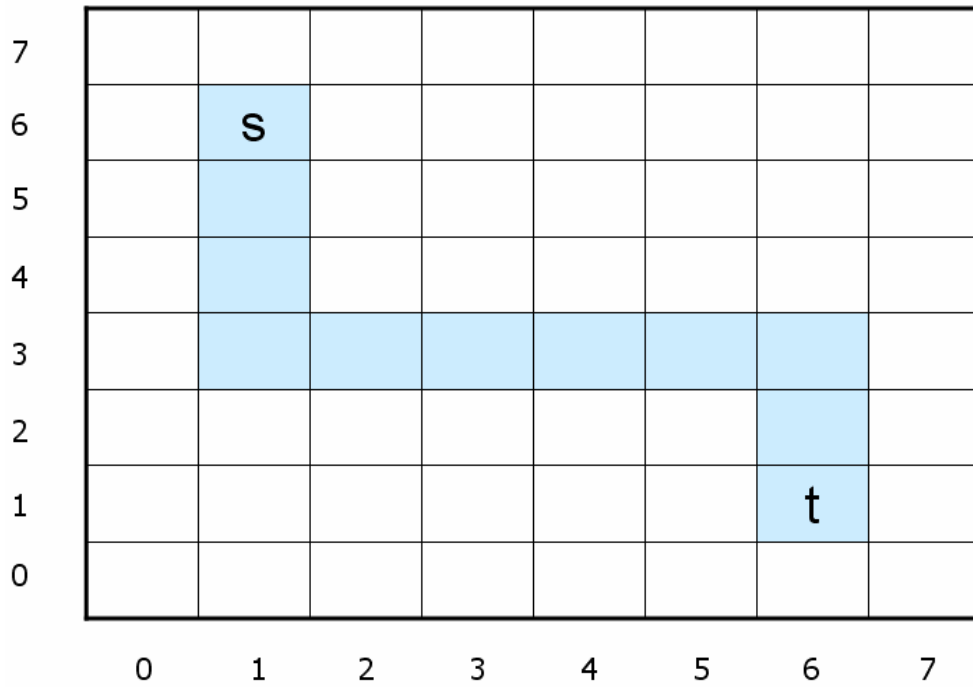


**Figure 4-4. Tile propagation in the active GCell**

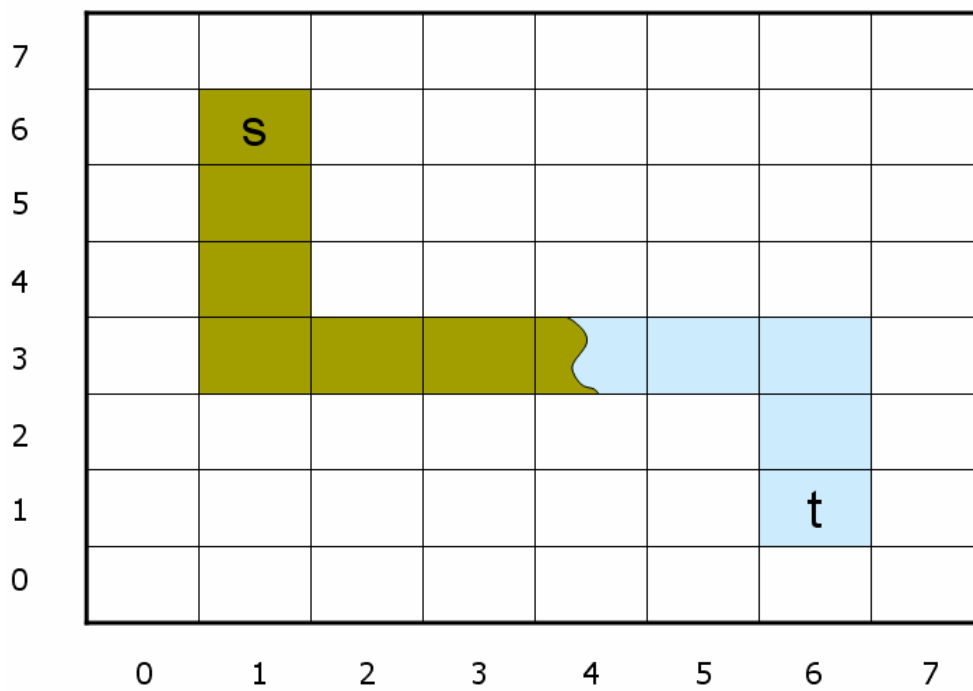
## 4.2 Extended Routing and GCell Restructuring and Rescheduling

In this section, we describe the solution to the problem when the router cannot find a feasible path along the active GCells. We use Figure 4-5 to illustrate the routing failure. Figure 4-5(a) shows the active GCells of a global routing from  $s$  to  $t$ . Figure 4-5(b) shows the propagation status, the darker GCells stand for the *visited GCells*. A visited GCell that does not contain a path passing through itself is called a *blocked GCell*. Therefore,  $GC(1,6)$ ,  $GC(1,5)$ ,  $GC(1,4)$ ,  $GC(1,3)$ ,  $GC(2,3)$ ,  $GC(3,3)$ , and  $GC(4,3)$  are visited GCells, and  $GC(4,3)$  is a blocked GCell. We can image that there is a wall blocking all the paths in  $GC(4,3)$  to propagate to next GCell,  $GC(5,3)$ . We propose two methods to solve this problem. The first is to expand the searching scope around the blocked GCell, called extended routing. The second is the GCells restructuring and rescheduling. Based on previous routing result, we restructure the internal edges of visited GCells and then reschedule a new set of active GCells by performing a new global routing.





(a)



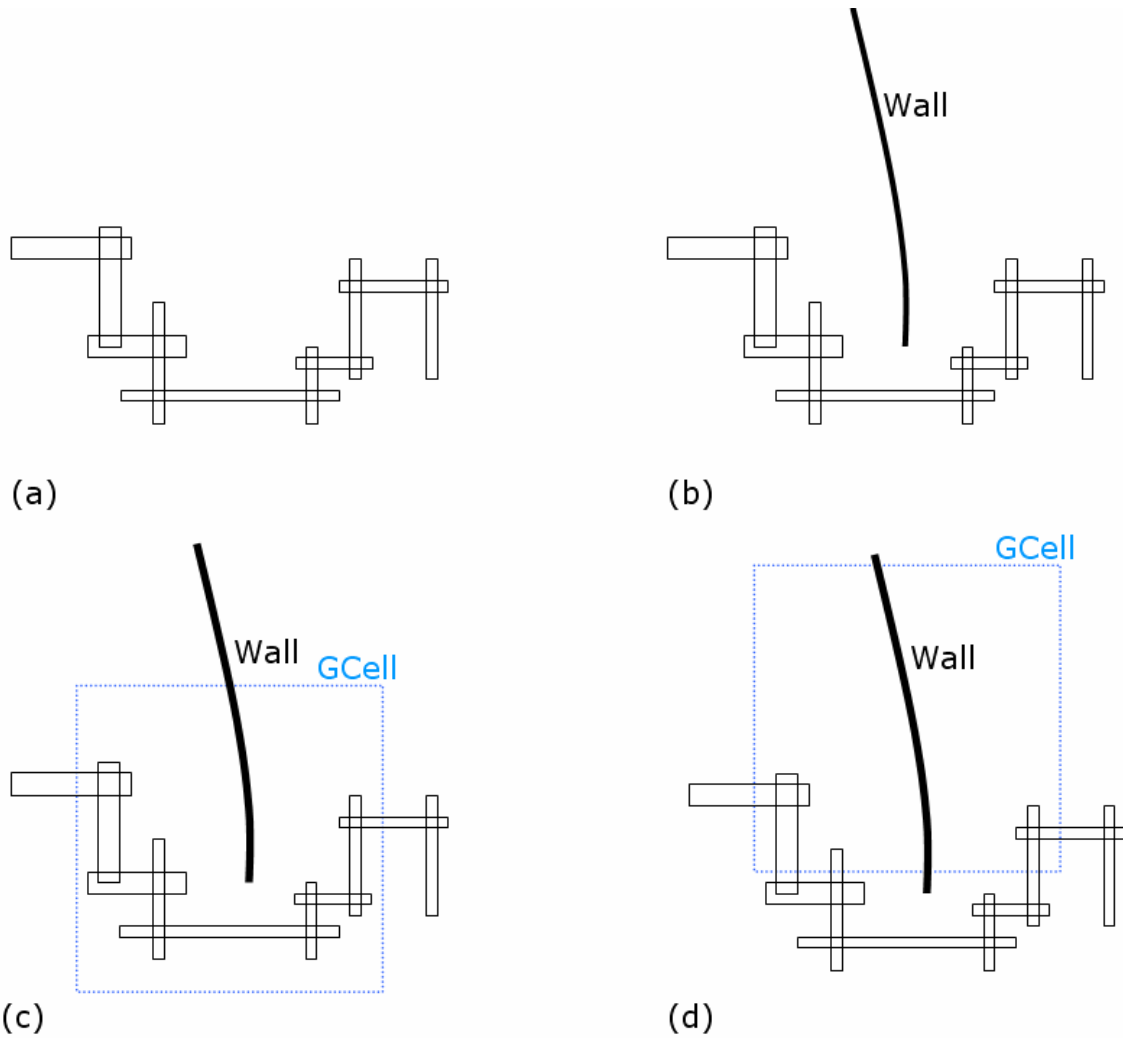
(b)

Figure 4-5. Tile propagation status

### 4.2.1 Extended Routing

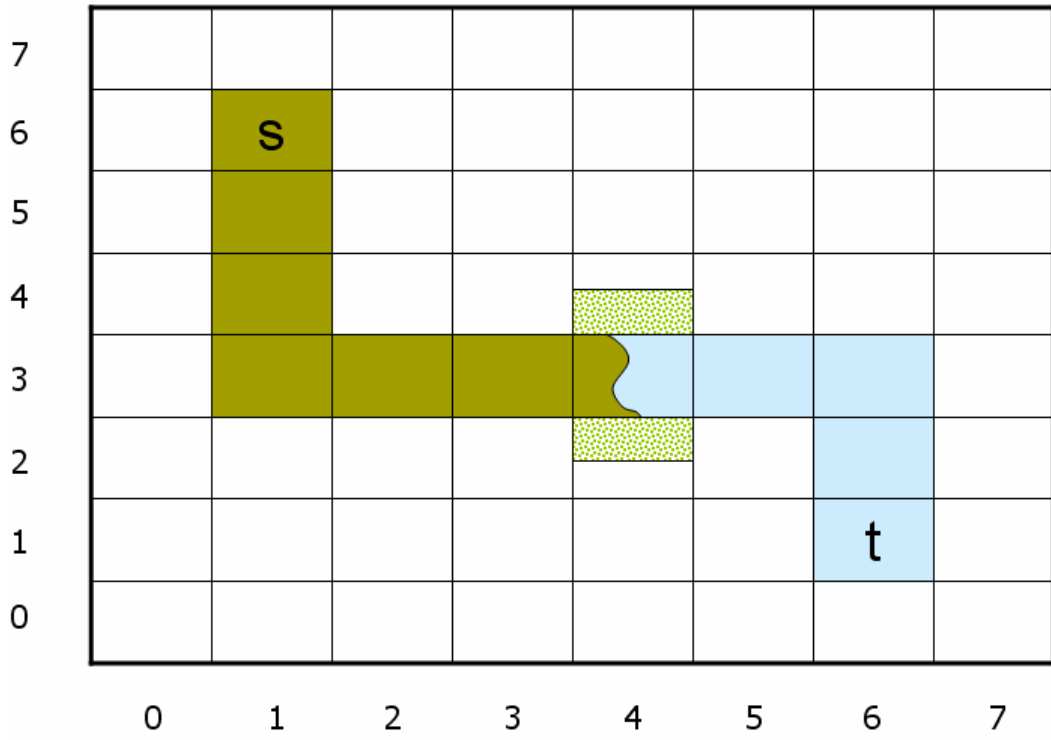
As mentioned in chapter 4.1, since there are lots of existing obstacles for ECO routing, the path is usually zigzag, as shown in Figure 4-6(a). Assume that there is a wall above the zigzag passage, as shown in Figure 4-6(b). If the active GCell is located as Figure 4-6(c), the tile propagation can pass through it. On the contrary, if the active GCell is located as Figure 4-6(d), it will become a blocked GCell. We can not prohibit such an infeasible partition since we do not know where the walls are before detailed routing.

Extended routing is to expand by a half of GCell width the searching scope around the blocked GCell. The tile propagation starts from the path nodes in the idle-path heaps of those idle GCell around the blocked GCell. For example,  $GC(4,3)$  is the blocked GCell in Figure 4-7(a), and the searching scope is expanded around  $GC(4,3)$ . The path nodes in the idle-path heaps of  $GC(4,4)$  and  $GC(4,2)$  will be popped up for further propagation. Extended routing increases the possibility to connect the two active GCells neighboring the blocked GCell. Figure 4-7(b) shows a successful connection between  $GC(3,3)$  and  $GC(5,3)$ .

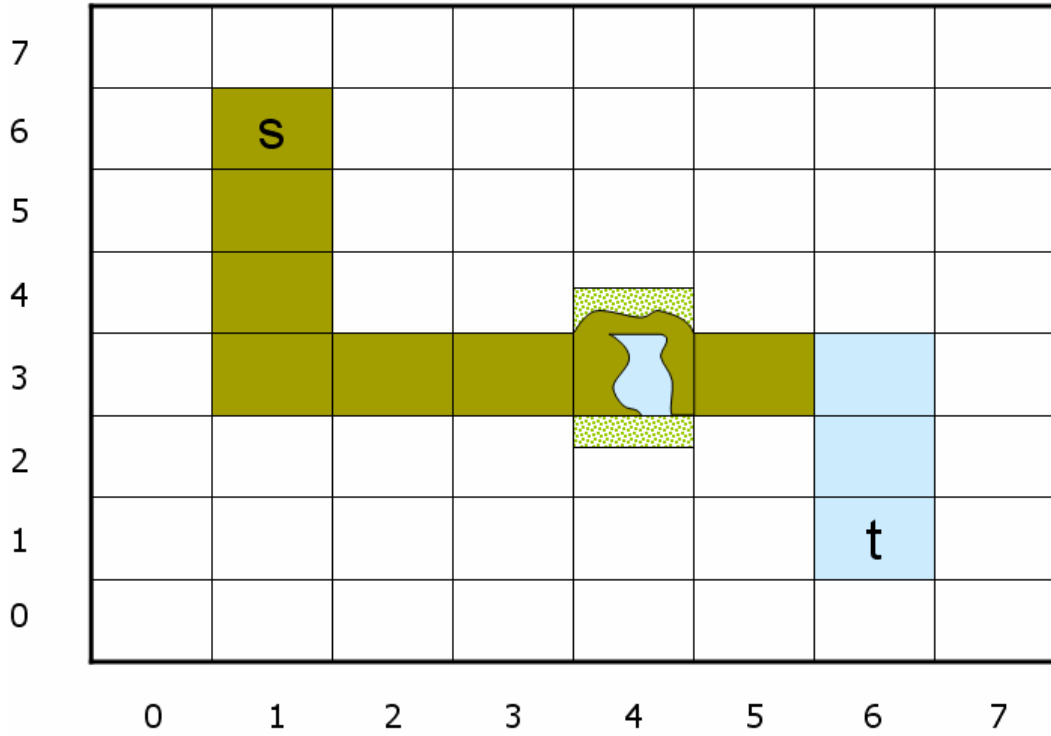


**Figure 4-6. GCell's location**





(a)



(b)

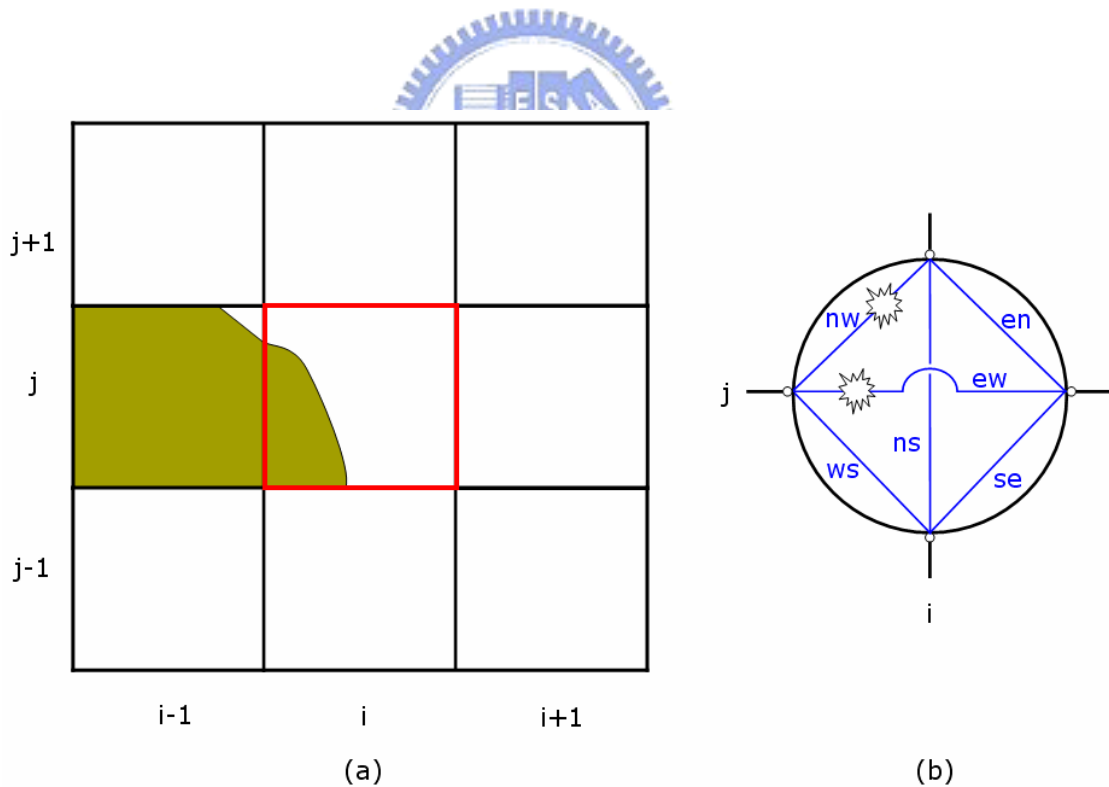
Figure 4-7. Extended Routing

## 4.2.2 GCell Restructuring and ReScheduling

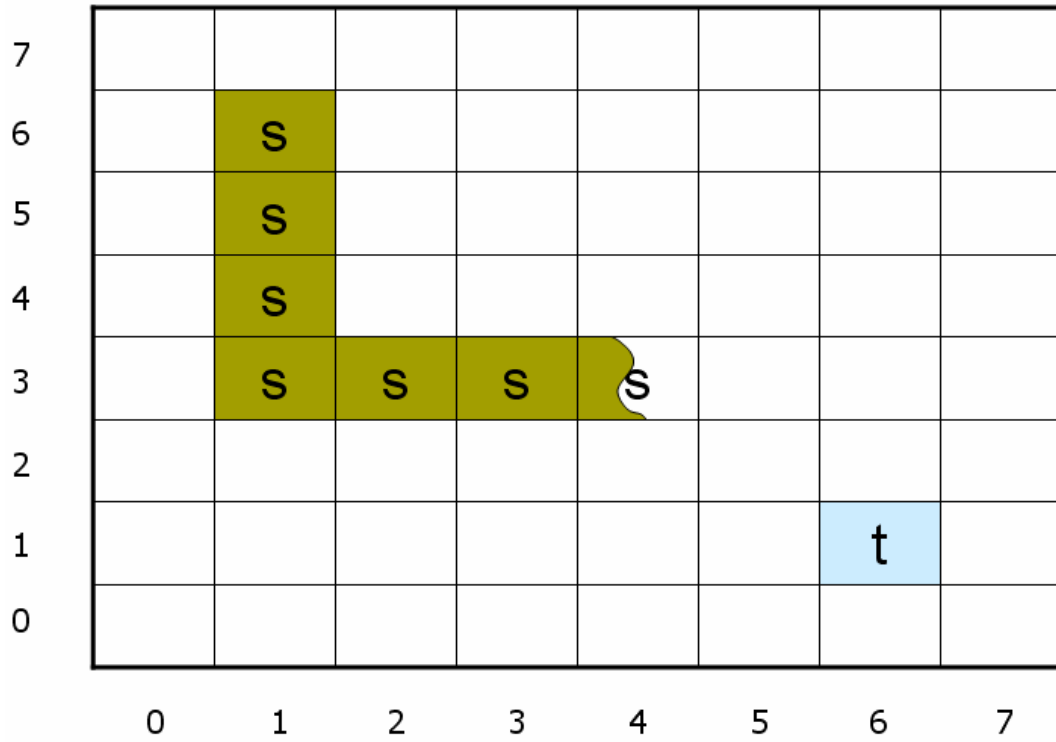
Whenever a blocked GCell appears, we apply extended routing to pass through the blocked GCell until the final feasible path is found or the extended routing fails. If extended routing still can not pass through the blocked GCell, we reschedule the active GCells to find another path. Before rescheduling the active GCells, we have to modify the internal edges' connectivity states of the active nodes to reflect the blocked information between GCells. In Figure 4-8(a),  $GC(i,j)$  is the blocked GCell. It is obvious that there is no path that passes through  $GC(i,j)$  and connects from  $GC(i-1,j)$  to  $GC(i+1,j)$  and  $GC(i,j+1)$ . It is very easy to determine the internal edges' connectivity states of the active nodes. Assume that we want to know the connectivity states of the internal edges of the node  $GC(i,j)$  and the routing direction through  $GC(i,j)$  is from left to right, we only have to check the idle-path heaps of  $GC(i,j+1)$  and  $GC(i,j-1)$ . If the idle-path heaps of  $GC(i,j-1)$  and  $GC(i,j+1)$  are empty, the connectivity states of edges  $nw$  and  $ws$  should be disconnected. Meanwhile, we can set the connectivity state of edge  $ew$  to be disconnected if  $GC(i,j)$  is a blocked GCell. Figure 4-8 shows an example of routing failure, where the connectivity states of edges  $nw$  and  $ew$  are set to be disconnected.

After restructuring the internal edges, a new global routing is performed using the visited GCells as the start vertexes, as shown in Figure 4-9(a). Based on the updated connectivity of internal edges, the result of the new global routing will not select the blocked channel. The new active GCells are idle GCells before, and those that are adjacent to old active GCells may have nonempty idle-path heap. We pop up these nonempty idle-path heaps to proceed new tile propagation. In Figure 4-9(b),  $GC(3,2)$ ,  $GC(3,1)$ ,  $GC(4,1)$ , and  $GC(5,1)$  are new active GCells, and  $GC(3,2)$ , which is adjacent to old active GCell,  $GC(3,3)$ , must have a nonempty idle-path heap.

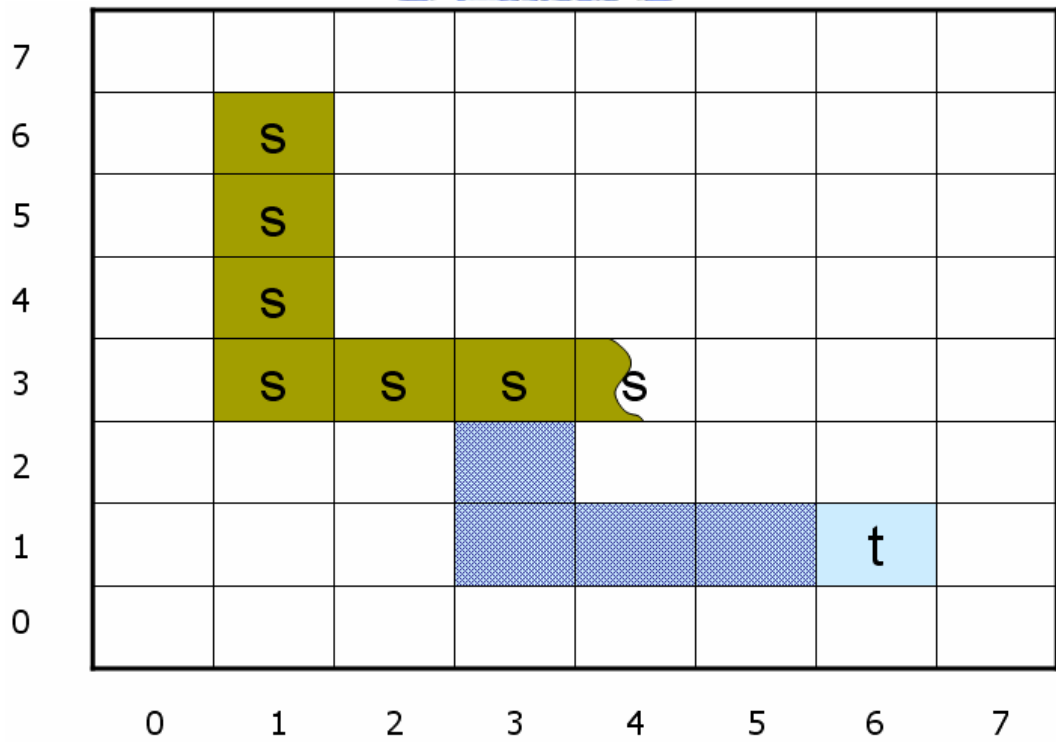
The GCell restructuring and rescheduling may be repeated several times during tile propagation. In Figure 4-9(c), tile propagation stops again at  $GC(4,1)$ . So we restructure and reschedule active GCells, as shown in Figure 4-9(d). The path nodes in the idle-path heaps of  $GC(4,2)$  may come from  $GC(4,3)$ ,  $GC(4,1)$ , and  $GC(3,2)$ , and they will be popped up for further propagation. It is worth to note that when we reschedule the active GCells, the number of the active GCells will increase. The worst case is to make all the GCell active and tile propagation will be over the whole tile plane. Therefore, we can guarantee to find a feasible solution if there exists such a solution. Moreover, even though the worst case happens, the routing speed will not drop off greatly because most already visited routing regions will not be visited again as a result of good routing resource estimation.



**Figure 4-8. GCell Restructuring**

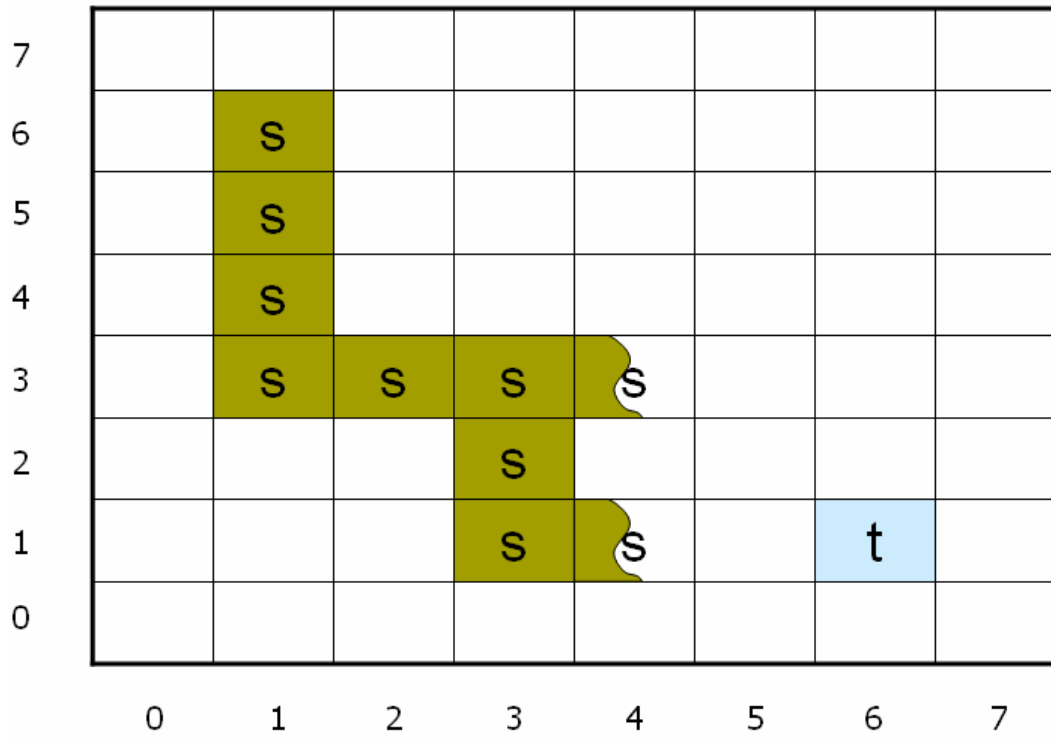


(a)

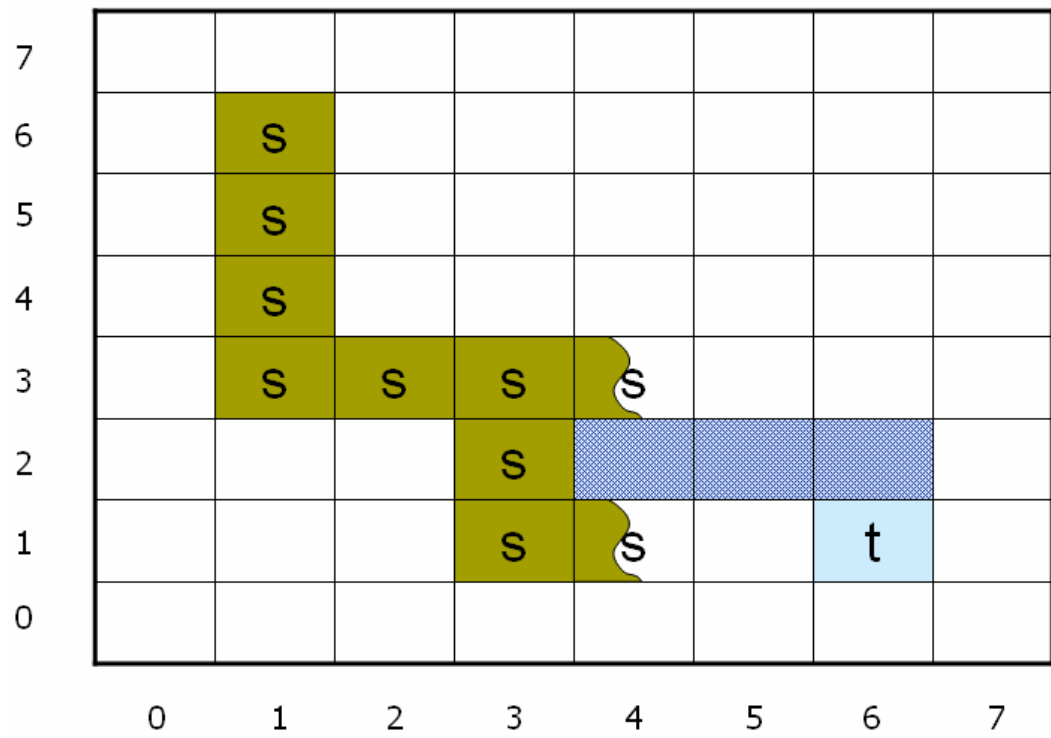


(b)

Figure 4-9. GCell ReScheduling



(c)



(d)

Figure 4-9. GCell ReScheduling

# Chapter 5

## Experimental Results

We have implemented a tile-based ECO router with all enhanced algorithms in this thesis using C++ language. We tested a test case from a real VLSI design on a 2.4GHz Pentium4 PC with 1GB Ram and performed p2p routings using *metal2* and *metal3* layers.

Table 5.1 shows the number of tiles of *metal2* and *metal3* layers in the corner stitching planes. The results without routing graph reduction are listed at the second column, and the third column is obtained after applying routing graph reduction. The number of tiles decreases by 57%. We list the pre-process time before routing in Table 5.2. The construction time of corner-stitching planes is listed at the second column and the time performing RGR is listed at the third column.

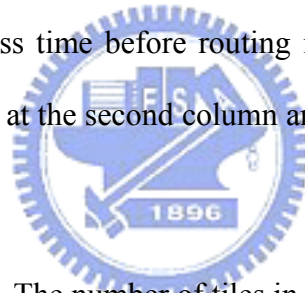


Table 5.1. The number of tiles in corner stitching planes

Layer	Origin		After RGR	
	#Block Tiles	#Space Tiles	# Block Tiles	#Space Tiles
Met2	1067179	973528	470325	380798
Met3	591120	541706	289144	218444
Total	3173533 ( $C_1$ )		1358711 ( $C_2$ )	
Reduction rate	0.571			

※ Reduction rate:  $(C_1 - C_2) / C_1$

Table 5.2. The pre-process time before routing.

Pre-process	CS Plane Construction ( $T_{cs}$ )	RGR ( $T_{rgr}$ )
Time (second)	21.656	5.889

We performed seven p2p routings using three methods to get the routing statistics, including the routing time (tile propagation time), wire length and the number of vias. In table 5.3, the second column is for pure tile propagation, and the third column for the case of applying routing graph reduction. When applying routing graph reduction, the routing time improve about 50%, as shown in column T1. Besides, it requires additional pre-process time. Hence, the total improve rate is about 40 % as shown in column T2. The wire length and the number of via are almost the same.

Table 5.4 shows the case of applying routing graph reduction and ECO global routing, and the right two items in the second column show how many times the extended routing and GCell restructuring and rescheduling are performed. Compared with pure tile-based router, the cost time decreases about 80-90%, as shown in column T3. However, the wire length increases by 3~30%, as shown in column W. The wire length is much longer for TEST5 because the start terminal is located on a very density region. Therefore, the global router is hard to find a good global path.

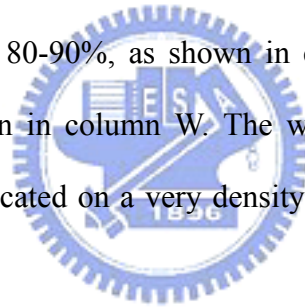


Table 5.3. Apply Routing Graph Reduction

Test name	Pure tile-based router			Apply RGR			T1(%)	T2(%)
	RT ( $T_a$ )	WL ( $W_a$ )	#Vias ( $V_a$ )	RT ( $T_b$ )	WL ( $W_b$ )	#Vias ( $V_b$ )		
TEST1	128.3	12035.07	480	65.8	12035.08	480	48.7	44.2
TEST2	82.6	11553.93	478	41.8	11553.96	478	49.5	42.3
TEST3	73.9	11399.45	394	38.3	11399.48	394	48.2	39.4
TEST4	65.3	9667.94	398	33.1	9667.97	398	49.3	40.3
TEST5	52.2	11194.86	508	26.5	11194.92	508	49.7	38.0
TEST6	121.5	12618.18	498	64.4	12618.21	498	47.0	42.2
TEST7	147.1	11732.99	502	75.6	11733.03	502	48.7	44.7

※  $T1 : (T_a - T_b) / T_a$ ,  $T2 : (T_a - (T_b + T_{rgr})) / T_a$

※  $RT$ : routing time(second),  $WL$ : wire length(um)

Table 5.4. Apply ECO Global Routing with Routing Graph Reduction.

Test name	Apply RGR +ECO Global Routing					T3(%)	W (%)	V(%)
	RT ( $T_c$ )	WL ( $W_c$ )	#Vias ( $V_c$ )	#ER	#GCRS			
TEST1	8.56	12497.55	184	0	0	88.7	3.8	-61.6
TEST2	4.62	12607.58	530	1	0	87.2	9.1	10.8
TEST3	4.28	12173.36	536	0	0	86.2	6.7	36.0
TEST4	4.62	10687.03	416	4	4	83.9	10.5	4.5
TEST5	5.29	14437.81	588	4	2	78.5	28.9	15.7
TEST6	5.56	13533.45	516	0	0	90.5	7.2	3.6
TEST7	6.92	13097.15	262	1	0	91.2	11.6	-47.8

※ The global routing partitions the layout to 28x28 GCells and each GCell's width is about 122 pitches.

※  $ER$ : Extended Routing,  $GCRS$ : GCell restructuring and rescheduling

※  $RT$ : routing time(second),  $WL$ : wire length(um)

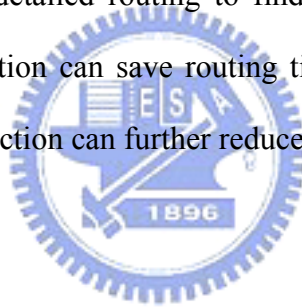
※  $T3 : (T_a - (T_c + T_{rgr})) / T_a$ ,  $W : (W_c - W_a) / W_a$ ,  $V : (V_c - V_a) / V_a$



# Chapter 6

## Conclusions

In this thesis we presented two methods to promote routing speed of the tile-based router. The first is routing graph reduction. By redundant tiles removal and neighbor tiles alignment, it diminishes tile fragmentation as well as reduces the routing time without sacrificing routing quality. The second is ECO global routing. It improves a lot the speed of the routing at little expense of routing quality. Also we proposed a routing flow, including extended routing and GCell restructuring and rescheduling, to incrementally expand ECO routing regions when global routing can not guide detailed routing to find a feasible path. Experimental results show that routing graph reduction can save routing time by about 40%, while ECO global routing with routing graph reduction can further reduce routing time up to 85% or so.



# Bibliography

- [1] J. Cong, L. He, C.-K. Koh, and P. Madden, "Performance optimization of VLSI interconnect layout," *Intergr. VLSI J.*, vol. 21, no. 1–2, pp. 1–94, Nov. 1996.
- [2] T. Ohtsuki, "Gridless routers—New wire routing algorithms based on computational geometry," in *Proc. Int. Conf. Circuits and Systems*, pp. 802809, May 1985.
- [3] K. L. Clarkson, S. Kapoor, and P. M. Vaidya, "Rectilinear shortest paths through polygonal obstacles in  $O(n(\log n))$  time," in *Proc. 3rd Annual Symp. Computational Geometry*, 1987, pp. 251–257.
- [4] Y. Wu, P. Widmayer, M. Schlag, and C. Wong, "Rectilinear shortest paths and minimum spanning trees in the presence of rectilinear obstacles," *IEEE Trans. Computers*, vol. C-36, no. 1, pp. 321-331, 1987.
- [5] S. Zheng, J.S. Lim, and S. Iyengar, "Finding obstacle-avoiding shortest paths using implicit connection graphs," *IEEE Trans. Computer-Aided Design*, vol. 15, no. 1, pp. 103-110, Jan. 1996.
- [6] J. Cong, J. Fang, and K. Khoo, "An implicit connection graph maze routing algorithm for ECO routing," in *Proc. Int. Conf. Computer-Aided Design*, pp. 163167, Nov. 1999.
- [7] J. Cong, J. Fang, and K. Khoo, "DUNE: A multilayer gridless routing system with wire plan-ning," in *Proc. Int. Symp. Physical Design*, Apr. 2000, pp. 1218.

- [8] J. Cong, J. Fang, and K. Khoo, "DUNE - A multilayer gridless routing system," *IEEE Trans. Computer-Aided Design*, vol. 20, no. 5, pp. 633-647, May. 2001.
- [9] M. Sato, J. Sakanaka, and T. Ohtsuki, "A fast line-search method based on a tile plane," in *IEEE Int. Symp. Circuits and Systems*, pp. 588-591, May 1987.
- [10] A. Margarino, A. Romano, A. De Gloria, F. Curatelli, and P. Antognetti, "A tile-expansion router," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 507-517, July 1987.
- [11] R. Eric Lunow, "A Channelless, Multilayer Router," in *25th ACM/IEEE Design Automation Conference*, pp. 667 - 671, 1988.
- [12] L.-C. Liu, H.-P. Tseng, and C. Sechen, "Chip-level area routing," in *Proc. Int. Symp. Physical Design*, pp. 197-204, Apr. 1998.
- [13] C. Tsai, S. Chen, and W. Feng, "An H-V Alternating Router," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 976-991, Aug. 1992.
- [14] J. Dion and L. M. Monier, "Contour: A tile-based gridless router," Western Research Laboratory, Palo Alto, CA, Research Report 95/3.
- [15] Zhaoyun Xing and Russell Kaog, "Shortest Path Search Using Tiles and Piecewise Linear Cost Propagation," *IEEE Trans. Computer-Aided Design*, vol. 21, no. 2, pp. 145-158, Feb. 2002.



[16] J. K. Ousterhout, “Corner Stitching: A data-structuring technique for VLSI layout tools,” *IEEE Trans. Computer-Aided Design*, vol. CAD-3, pp.87–100, Jan. 1984.

[17] Preparata Franco P. & Shamos Michael Ian, “Computational geometry : an introduction”, Springer-Verlag, New York ,1985

