國 立 交 通 大 學

電信工程學系

碩 士 論 文

具備導線推擠能力的高效能多餘貫穿點插入方法

An Efficient Redundant Via Insertion Method with Wire Pushing Capability

研究生：李焯基

指導教授：李育民 教授

中 華 民 國 九 十 七 年 六 月

具備導線推擠能力的高效能多餘貫穿點插入方法

學生:李焯基　　　　　　　　　　　　　　指導教授:李育民 博士

國立交通大學電信工程學系碩士班

摘　　要

隨著製程進入奈米時代，能有效的微影以及得到良好的製造程序變得愈來愈困難。在積體電路實體層設計上導致良率下降的其中一個主要因素為貫穿點(via)的損毀。因此，得到一個良好的控制基制去改善貫穿點的製造良率以及可靠度，是可製造設計(DFM)的一個重要課題。

插入多餘貫穿點是一個被證實有效並且被廣泛推薦來增加貫穿點良率以及可靠度的方法。在這篇論文，我們發展一個在佈線後能高效能插入多餘貫穿點的演算法來增加良率。我們把插入多餘貫穿點的問題轉換成混合二部碰撞圖形的匹配問題，並提出一個新穎的啟發式最小加權匹配演算法去解決此問題。此方法除了插入多餘貫穿點於存活貫穿點(Alive via)，還運用導線推擠的概念去保護荒廢貫穿點(Dead via)，方法是把荒廢貫穿點鄰近的導線往空置的空間移開使能在其旁加上多餘貫穿點。實際結果證明我們的方法可以獲得一個高的多餘貫穿點插入率並能有效地執行。

An Efficient Redundant Via Insertion Method with Wire Pushing Capability

Student: Cheok-Kei Lei Advisor: Dr. Yu-Min Lee

Department of Communication Engineering
National Chiao Tung University

ABSTRACT

As the process technologies advancing to nanometer epochs, the lithography and manufacture procedures turn out to be more difficult. Via defect is one major source of yield loss during the physical design stage. Therefore, requiring a good control for via failure to improve via yield and reliability is one of the most important issues in design for manufacturability (DFM).

Redundant via insertion is a verified efficient and widely recommended method to enhance via yield and reliability. The purpose of this thesis is to develop an efficient method to insert redundant vias in the post-routing stage. We transform the redundant via insertion problem into a mixed bipartite-conflict (MBC) graph matching problem, and present a novel heuristic minimum weighted matching algorithm (HMWM) to solve it. The proposed method, besides inserting redundant vias for alive vias, also protects the dead vias by applying wire pushing capability which shifts wires into the empty space and adds redundant vias next to dead vias. Experimental results show that our method obtains a high redundant via insertion rate and perform effectively.

誌　謝

這篇論文能順利完成，首先要衷心感謝指導老師李育民老師，在碩士生涯的兩年裡，提供了良好的學長環境，還有在研究上的幫助指導以及生活上的各種叮嚀關懷，學生對老師為我所付出的一切銘記於心，並且在我的成長與學習過程中佔了一個相當重要的位置。

研究最重要是互相討論，教學相長，在此相當感謝江柏毅學長對論文所提供的建議及幫忙，使論文更趨完備。

在實驗室，感謝培育學長的知識經驗傳承，以及同窗伙伴斯安跟我一齊打拼，互相支持鼓勵。還有感謝宗祐、阿文學弟日常生活上帶給我的歡樂，以及至鴻、志康、國富、阿忠、佳鴻、庚達、俊諺、懷中平日的關心與幫忙。謝謝以上同伴一路相伴成長，帶給我很多歡樂。

同時要感謝我的家人自小對我諄諄教誨，提供一切所需栽培我成才，使我能無後顧之憂、專心投入於研究上，在學習的路上，你們一直都是我的精神支柱，期間時時刻刻想到的，就是努力唸書使能順利畢業，未來有所回報，讓你們能快樂的安享晚年。最後要感謝澳門的所有朋友對我的支持與照顧，特別是裕威、家威及志威。

總之衷心感覺以上所提及以及曾經幫助、關心與支持我的朋友，願我未來能有所成就，回饋你們，在此把論文獻給你們，共享這份成果與喜悅。

Contents

1 Introduction **1**
1.1 Introduction . . . 1
1.2 Our Contributions . . . 3
1.3 Organization of the Thesis . . . 4

2 Preliminaries and Literatures Overview **5**
2.1 Basic Concept of Post-routing Redundant Via Insertion . . . 5
2.1.1 Post-routing Redundant Via Insertion . . . 5
2.1.2 On-track and Off-track Redundant Via . . . 5
2.1.3 The Categories of Single Via . . . 6
2.1.4 Stack Via . . . 7
2.2 Previous Works Related to Redundant Via Insertion . . . 7

3 Post-routing Redundant Via Insertion with Wire Pushing Capability **10**
3.1 The Treatment of Stack Via . . . 10
3.2 Wire Pushing Capability . . . 11
3.3 MBC Graph Matching Problem . . . 13
3.4 Redundant Via Insertion Method with Wire Pushing Capability (RVI^{f}A-WP) . . . 17
3.4.1 Redundant Via Insertion for Alive Vias (RVI^{f}A) . . . 18
3.4.2 Dead Via Protection by using Wire Pushing Capability (WP) . . . 23
3.4.3 Runtime Complexity Analysis of HMWM Algorithm . . . 27
3.4.4 Speed Up . . . 29

4 Experimental Results **30**

5 Conclusion **41**

List of Figures

1.1 Vias in chip 1
1.2 Single via failure [2] 2

2.1 Redundant vias in different directions. (a) On-track redundant via. (b) Off-track redundant via. 6
2.2 The different categories of single via. 7
2.3 Stack via connections in 0.35-mm CMOS technology interconnect [16] 8

3.1 Redundant via insertion of stack via structure 11
3.2 An example of wire pushing capability. (a) A portion of original routed circuit. (b) Two redundant via candidates after pushing wire segments. Here, the wire shifted degree of the top redundant via is 1, and the wire shifted degree of the bottom redundant via is 2. 12
3.3 Two feasible cases without violating the design rule. 14
3.4 An example of mixed bipartite-conflict graph. (a) A portion of routed circuit. (b) The via-candidate bipartite graph. (c) The candidate relative graph. (d) The mixed bipartite-conflict graph. 16
3.5 Illustration of the difference between the mixed bipartite-conflict graph and the bipartite graph 17
3.6 The flowchart of the proposed RVI^{f}A-WP method. 18
3.7 Illustion of the conflict between two redundant via candidates for the alive and dead vias. 20
3.8 Algorithm of HMWM. 21
3.9 An example for illustrating the RVI^{f}A procedure. 24
3.10 Illusion of a conflict between two pushing wire segments 26
3.11 An example for illustrating the WP procedure. 28

4.1 (a) A portion of original routing of circuit S38584 (b) A portion of wire pushing result of circuit S38584 34
4.2 (a) The distribution of dead vias of circuit Struct. (b) The distribution of dead vias after wire pushing of circuit Struct. 35
4.3 (a) The distribution of dead vias of circuit Mcc2. (b) The distribution of dead vias after wire pushing of circuit Mcc2. 36
4.4 (a) The distribution of dead vias of circuit S38584. (b) The distribution of dead vias after wire pushing of circuit S38584. 37
4.5 (a) The original routing result of circuit Struct. (b) The distribution of pushing wire segments of circuit Struct. 38
4.6 (a) The original routing result of circuit Mcc2. (b) The distribution of pushing wire segments of circuit Mcc2. 39

4.7 (a) The original routing result of circuit S38584. (b) The distribution of pushing wire segments of circuit S38584. 40

List of Tables

4.1 Comparison for the proposed RVI^{f}A procedure with TDVI [14]. 33
4.2 Experimental results of the proposed RVI^{f}A-WP method. 33

Chapter 1

Introduction

1.1 Introduction

As the VLSI technology scales into the deep sub-micron and nano-meter region, the number of transistors and logic gates speedily increases in a chip. Nowadays, modern chips may have six or more metal layers and over 4,000,000 vias [1]. The vias are used to connect wires on different metal layers and play a very important role in the integrated circuit (IC) design. Fig. 1.1 shows the vias with local and global interconnects.

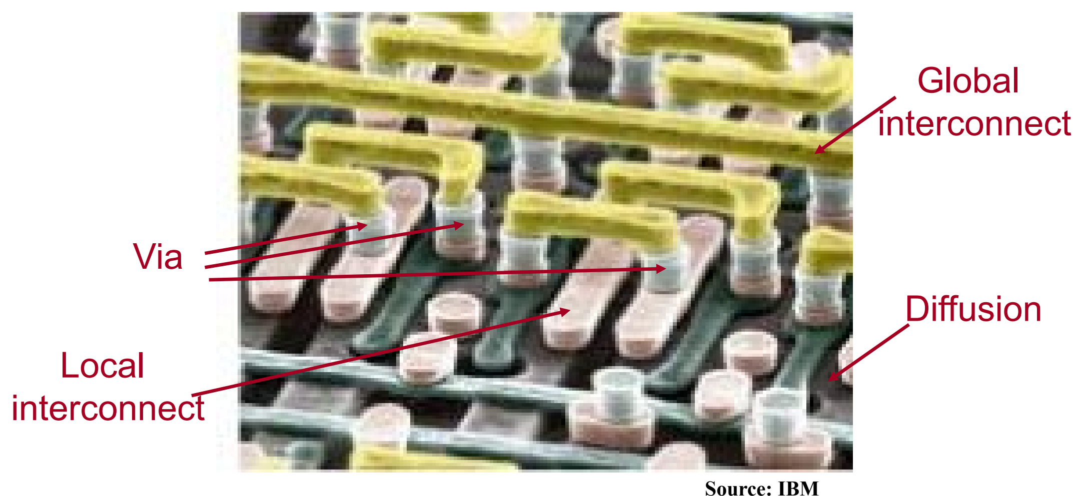


Fig. 1.1: Vias in chip

The via may be partially or completely failed [2] due to the random defects in a manufacturing process, electro-migration and thermal stress. If a via is partially blocked, its substantially increasing equivalent resistance and capacitance will degrade the circuit performance and cause

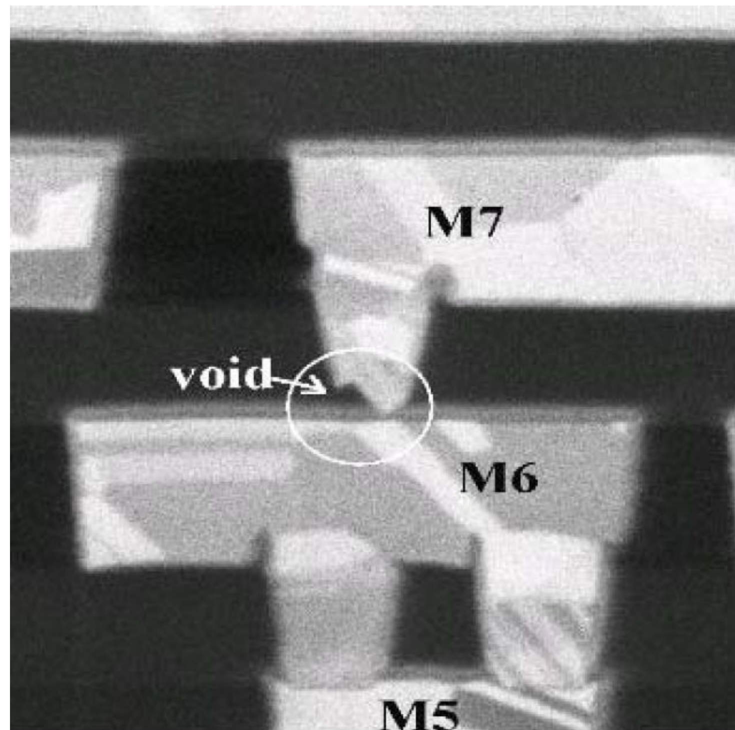


Fig. 1.2: Single via failure [2]

the reliability issue of the circuit. On the other hand, the completely failed via will break the net, cause the functional failure of logic gates and make the entire design fail. Fig. 1.2 shows the single via failure. Yield will lose seriously of defects because of these via failures. Therefore, a good control for via failure to improve the yield is one of the most important issues in design for manufacturability (DFM). Inserting a redundant via adjacent to a single via without causing any design rule violations is a valid and recommended method to improve the via yield and reliability [3, 4, 5, 6]. The redundant via can be viewed as a backup of the single via, and it makes the via failure to be tolerated. After inserting a redundant via, the failure rate of double vias is quite smaller than the single via's. It has been indicated that a single via fails 10X-100X more often than the double vias [7]. Many commercial EDA tools have already added the redundant via insertion function to their physical design flow.

Generally, the redundant via insertion technique can be performed in different stages such as the routing stage and the post-routing stage. To perform the redundant via insertion in the routing stage, the double-via insertion rate can be improved compared with in the post-routing stage. However, the inserted redundant vias might lead some non-routing nets to be unroutable

and degrade the routability, it also makes the routing to be more complicated. In contrast, inserting redundant vias in the post-routing stage is easy to perform and several existing methods have taken many effects such as the timing constraints, routable nets with the minimum die area and the antenna rule into account in the routing stage. The tools EYE/PEYE [8] insert redundant vias next to single vias in the post routing stage. Authors in [9], [10] and [11] also consider the redundant via insertion in the post-routing stage. If we consider the redundant via insertion in both the routing and the post-routing stages, it becomes a trade-off between routability and redundant via insertion rate. Neverless, decreasing the number of critical and dead vias causes some non-routing nets to make detours and generate more vias. Methods proposed in [12], [13] and [14] consider redundant via insertion in both the routing and the post-routing stages.

Besides, wire spreading [8] is used to reduce the critical area between the interconnect for increasing wire yield and reliability. Its basic idea is to spread wires into the empty space for reducing the probability of a defect particle shorting two neighboring wires. The tools EYE/PEYE [8] reduce the critical area to avoid shorts between wires of the layout by using wire spreading technique. This concept of wire spreading can be utilized to insert redundant vias next to dead vias as proposed in [15].

1.2 Our Contributions

In this work, we develop an efficient algorithm with wire pushing capability to insert redundant vias for both alive and dead vias in the post-routing stage. Firstly, the mixed bipartite-conflict (MBC) graph is defined by mixing the bipartite and conflict graphs, and a weight of each edge in the MBC graph is suitably assigned. Then, the redundant via insertion problem is formulated as a defined MBC graph matching problem. After that, an effective heuristic minimum weighted matching (HMWM) algorithm is developed to find the matching of MBC graph. Finally, we utilize the above procedure and the wire pushing technique to develop an efficient redundant via insertion method for alive and dead vias. Experimental results show that the average insertion rate of alive vias is 99.54% with a short run time, and the technique of wire pushing can achieve 54.41% insertion rate for the dead vias in average.

1.3 Organization of the Thesis

The rest of this thesis is organized as follows. Chapter 2 introduces the basic concept of redundant via insertion. At the same time, we also review some related literatures. In chapter 3, we first present the treatment of stack via and the concept of wire pushing capability and how to utilize this technique for inserting the redundant via adjacent to the dead via ,then second present the detail of our post-routing redundant via insertion method with wire pushing capability (RVI^{f}A-WP). Finally, the experimental results for the comparison of post-routing redundant via insertion between our algorithm and TDVI [14] are presented in chapter 4. The experimental results of wire pushing capability are also shown in chapter 4. Finally, a brief conclusion is given in chapter 5.

Chapter 2

Preliminaries and Literatures Overview

2.1 Basic Concept of Post-routing Redundant Via Insertion

In this chapter, the basic concepts of redundant via insertion is presented, and some previous related researches are reviewed.

2.1.1 Post-routing Redundant Via Insertion

The redundant via and the post-routing redundant via insertion problem are defined as follows.

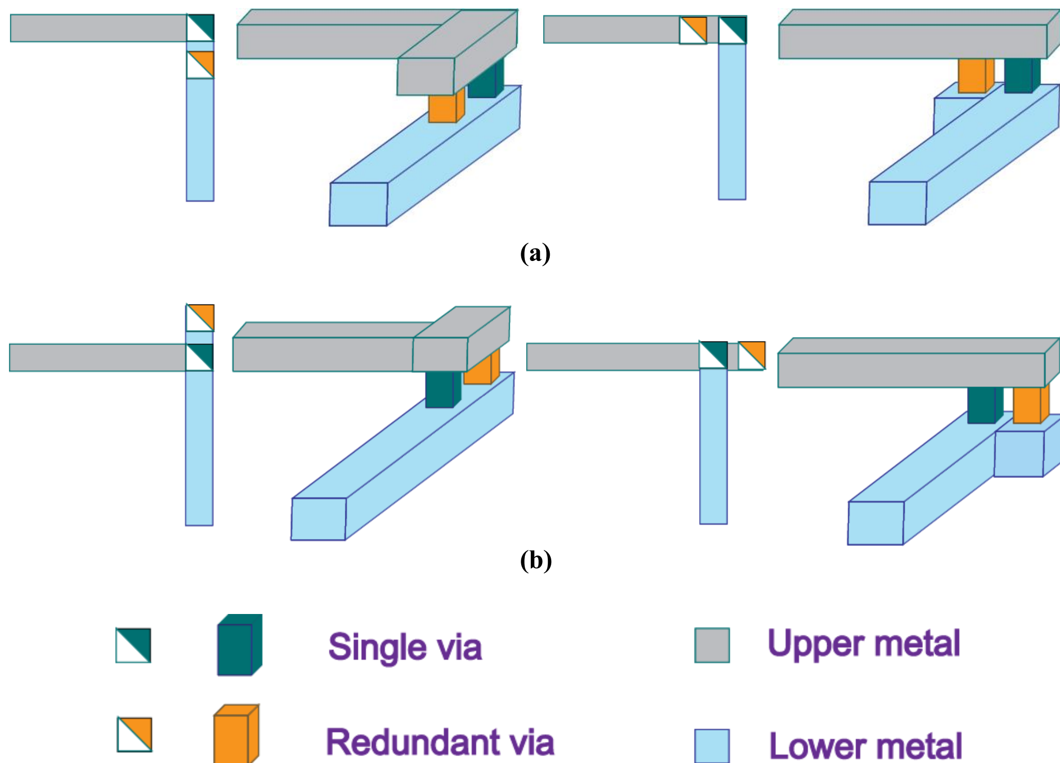
Redundant via: *Redundant via is a backup of single via; the process of redundant via insertion is to insert a redundant via adjacent to a single via as long as it does not cause any design rule violation. The name of the single via and its adjacent redundant via is called double via.*

Post-routing redundant via insertion problem: *Given a routed circuit design and technology design rules, the post-routing redundant via insertion problem is to insert an extra via next to a single via without violating any design rule, and try to protect single vias as many as possible. If inserting an extra via is not allowed, single via will keep unchanged. Post-routing redundant via insertion process is equal to replacing single via with double via.*

2.1.2 On-track and Off-track Redundant Via

Basically, redundant via is divided into two main types: on-track and off-track redundant via. On-track redundant via means that the second via is inserted onto the track of the original signal net as shown in Fig. 2.1(a) . On the other hand, off-track redundant via means that the second via is inserted with overhanging extra metal wire of the single via as shown in Fig. 2.1(b). On-

track redundant via is more critical than off-track redundant via since on-track redundant via takes less routing resource and has better electrical properties.



1896

Fig. 2.1: Redundant vias in different directions. (a) On-track redundant via. (b) Off-track redundant via.

2.1.3 The Categories of Single Via

According to the definition of [14], single via has three different categories according to the number of its redundant via candidates in the post layout. A redundant via candidate is a position where a redundant via can be inserted next to a single via without any design rule violation. A single via is called dead via if it does not have any redundant via candidate, whereas a via is called alive via if it has at least one redundant via candidate. Furthermore, an alive via is also called critical via if it has only one redundant via candidate. Fig. 2.2 illustrates the different via categories described above. After performing the redundant via insertion, if a redundant via can be inserted next to the original single via, the single via is protected. Otherwise, it is dead. How to effectively insert redundant vias for protecting critical and dead vias to get a higher

insertion rate is a practical issue of redundant via insertion in the post routing stage.

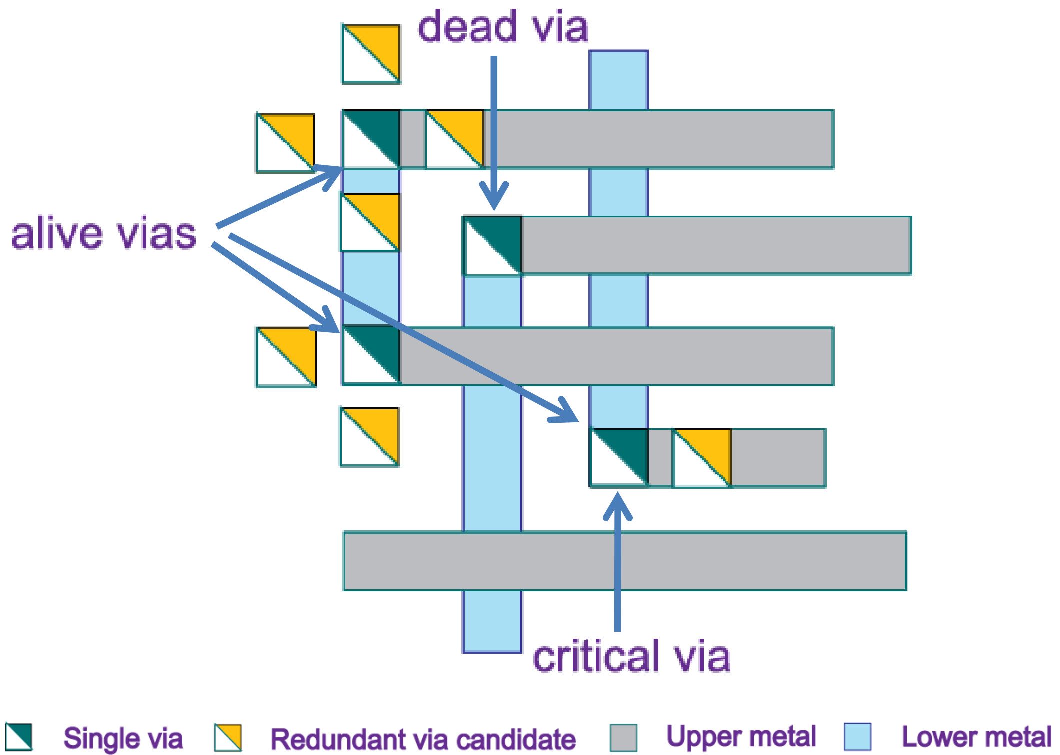


Fig. 2.2: The different categories of single via.

2.1.4 Stack Via

Chen et al. [14] reported a special via structure called stack via. It consists of at least two single vias which stack vertically. Fig. 2.3 shows this special structure in the interconnect of the chip.

2.2 Previous Works Related to Redundant Via Insertion

The first work considering the redundant via insertion for the yield improvement in the detailed maze routing stage was proposed by Xu et. al. [12]. They utilized a Lagrangian relaxation approach to insert redundant vias; however, the computational complexity of their approach is very high. Yao st. al. [13] minimized the number of via usage and considered redundant vias planning in the routing stage to improve the yield, but some routed wire segments might violate the antenna rules because of enabling some wire segments to be longer for reducing the number of vias in the detailed routing stage. Thus, it might need extra vias to be added in the

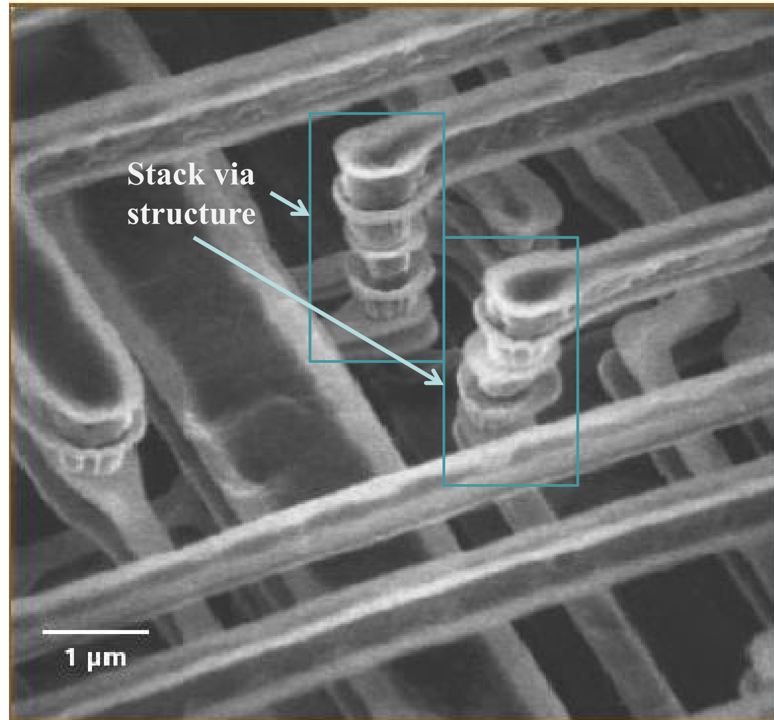


Fig. 2.3: Stack via connections in 0.35-mm CMOS technology interconnect [16]

circuit design to solve this antenna problem. Luo et. al. [11] presented a redundant via insertion method based on a novel geotopological platform. The post-routing layout is extracted to a geotopological layout, and then the insertion is performed one by one. This approach changes the routing results to obtain a high insertion rate.

Lee et. al. [9, 10] formulated the redundant via insertion problem as a maximum independent set (MIS) problem and considered redundant via insertion in the post-routing stage. Since the MIS is a NP–complete problem, they proposed an efficient model and a heuristic algorithm to solve it.

Recently, Chen et. al. [14] presented a novel full-chip gridless routing system considering the via number minimization in the global routing stage and the redundant via planning in the detailed routing stage. However, in order to decrease critical and dead vias, the count of vias slightly increases. They proposed an algorithm called TDVI based on the bipartite graph matching formulation to achieve a high redundant via insertion rate in the post-routing stage. Moreover, stack via is treated as an unit via in the insertion process, this treatment causes the quality of solution might be degraded. A simple example will be presented in subsection 3.1 to

explain this issue. Besides, as the number of routing layers for the layout is more than 3, their two-stage double-via insertion (TDVI) algorithm partitions the original layout into several sub-layouts which include three routing layers at most. After that, they first solve the lower critical sub-layout by using the maximum bipartite matching. Hence, the insertion solution quality of the TDVI algorithm is degraded due to this heuristic. They also formulated the redundant via insertion problem as a minimum weighted bipartite matching problem and gave higher priority to on-track redundant via candidates and stack via redundant via candidates for insertion. K. McCullen [17] presented a dynamic method by applying wire spreading technique [8] that shifts original wires to create space for inserting more redundant vias under the restricted topology layout. Although the method can achieve a higher insertion rate, the authors restrict the problem to a 1-D problem and the runtime efficiency is not very well.

Chapter 3

Post-routing Redundant Via Insertion with Wire Pushing Capability

In this chapter, the treatment of stack via and the basic concept of wire pushing capability are introduced first. Then, the post-routing redundant via insertion problem is formulated as a graph matching problem which we call it the *MBC graph matching problem*. After formulating the problem, the details of the proposed post-routing redundant via insertion method with wire pushing capability (RVI^{f}A) are presented.

3.1 The Treatment of Stack Via

To solve the counterexample shown in [9] by the method TDVI presented in [13], authors in [14] transformed the redundant via insertion problem to the maximum bipartite matching problem with up to three routing layer, and treated a stack via as one unit via. However, we find that this treatment will loss the quality of the final insertion solution. For example, Fig. 3.1(a) shows two different nets with a single via ($V3$) and a stack via which consists of two single vias ($V1$ and $V2$), respectvely. Fig. 3.1(b) indicates all possible redundant via candidates of these single vias ($RB1$ and $RR1$ for $V1$, $RL2$ and $RR2$ for $V2$, and $RL3$ for V3). Assuming that the redundant via candidates $RR1$ and $RL3$, $RR2$ and $RL3$ cannot exist simultaneously because their vertical and horizontal conflict violate the design rule. Because $V1$ and $V2$ are stacked, we need to merge the same side feasible redundant via candidates $RR1$ and $RR2$ into one candidate as shown in Fig. 3.1(c). Under this constraint, it makes some available redundant via candidates missing such as $RB1$ and $RL2$. Fig. 3.1(d) shows the insertion solution after performing the minimum weighted bipartite matching. This result is not the optimal one in this example, and

the optimal solution is shown in Fig. 3.1(e). In conclusion, to firstly protect stack via, single via which neighbors stack via may not be protected when its redundant via candidate has conflict with stack via's redundant via candidate. Therefore, in our approach, each single via is treated as an unit via and the insertion priority of their redundant via candidates are equitable if we do not consider on-track preference.

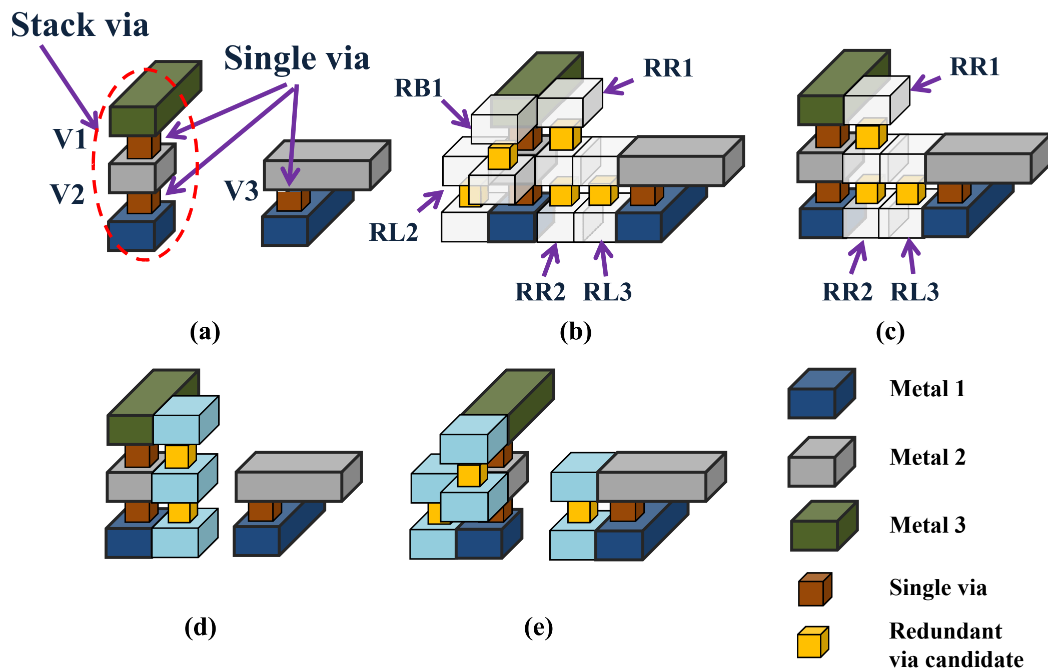


Fig. 3.1: Redundant via insertion of stack via structure

3.2 Wire Pushing Capability

A conventional method cannot insert redundant via next to a dead via if there is not enough empty space at the surroundings of it. In order to achieve a higher redundant via insertion rate, the redundant via insertion with layout pushing capability was proposed in [15]. Its basic idea is to push wire segments away the dead via without introducing any design rule and connectivity violations to allocate enough space for inserting a redundant via adjacent to this dead via. This is a practical method to protect dead vias in the post-routing stage, and some commercial tools

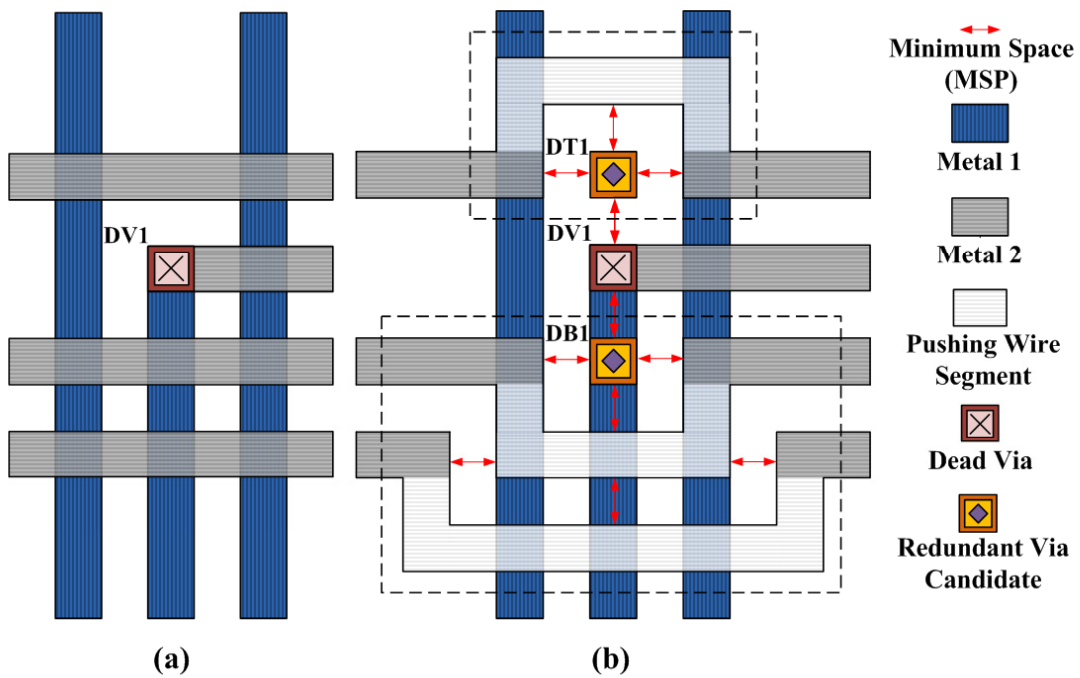


Fig. 3.2: An example of wire pushing capability. (a) A portion of original routed circuit. (b) Two redundant via candidates after pushing wire segments. Here, the wire shifted degree of the top redundant via is 1, and the wire shifted degree of the bottom redundant via is 2.

are disposed to develop this technique for improving the yield of dead vias. A simple example is shown in Fig. 3.2 to illustrate it. Fig. 3.2(a) is a portion of original routed circuit, and two illustrated redundant via candidates after pushing wire segments are shown in Fig. 3.2(b). The problem formulation of wire pushing can be described as follows.

Basically, given a routed circuit and process design rules, we should determine how to push wire segments around the dead vias to obtain enough space for inserting vias. After pushing the wire segments, the modified layout must maintain circuit characteristics, and the chip area must keep unchanged.

To find redundant via candidates for dead vias by utilizing the wire spreading capability, a searching region is predefined for finding a redundant via candidate of the dead via. The use of searching region is similar to the DRW[1] proposed in [9]. The searching region is a bounding rectangular box, and its size is according to the minimum space (MSP) and the maximum wire shifted degree. Here, the MSP is the minimum space design rule of metal and via in the same

[1]DRW is a bounding rectangular box that is used to find redundant via candidates for alive vias.

layer, and the maximum wire shifted degree is the extended level of wires allowed to be pushed. When the movement of wire segments in the searching region is legal, a redundant via candidate of the dead via and the minimum shifted degree of its adjacent wires are obtained. Fig. 3.2(b) illustrates the result of finding the redundant via candidates for a dead via in its searching region with the maximum wire shifted degree being 3 for each metal layer. Two redundant via candidates are presented in Fig. 3.2(b). One wire shifted degree is 1, and the other is 2.

The searching region is also utilized to find the conflicts– that is to say, the design rule is violated if some redundant via candidates of dead via and the redundant via candidates of single via exist simultaneously. Furthermore, single vias, pins and instance pins cannot exist on the pushing wire segments of the searching region.

The experimental results presented in chapter 4 demonstrate that the wire spreading technique can achieve average insertion rate to be 54.41% for the dead vias in the benchmark circuits, and the movement of pushing wires in the searching region is relatively small compared with the original layout. Hence, the impact of circuit timing is negligible.

3.3 MBC Graph Matching Problem

In this section, the post-routing redundant via insertion problem is formulated as a graph matching problem. Firstly, we give the definitions of several related graphs. Then, the graph matching problem which we call it the "*MBC graph matching problem*" is defined.

Definition 1 (Via-candidate bipartite graph) *A via-candidate bipartite graph* $G_v = (V \cup R, E_v)$ *is an undirected bipartite graph whose vertices are composed of two independent sets* V *and* R*.* V *is the set of single vias in the circuit, and it is named to be a single via set.* R *is the set whose vertices are the redundant via candidates of single vias, and it is called as a redundant via candidate set.* E_v *is the edge set. For each* $v \in V$ *and* $r \in R$*, there exists an edge* $e(v, r) \in E_v$ *if* r *is a redundant via candidate of* v*.* ■

Definition 2 (Candidate relative graph) *A candidate relative graph* $G_c = (R, E_c)$ *is an undirected conflict graph whose vertex set is the redundant via candidate set* R*. For each* $r_1 \in R$ *and* $r_2 \in R$*, there exists an edge* $e(r_1, r_2) \in E_c$ *if the design rule will be violated for simultaneously*

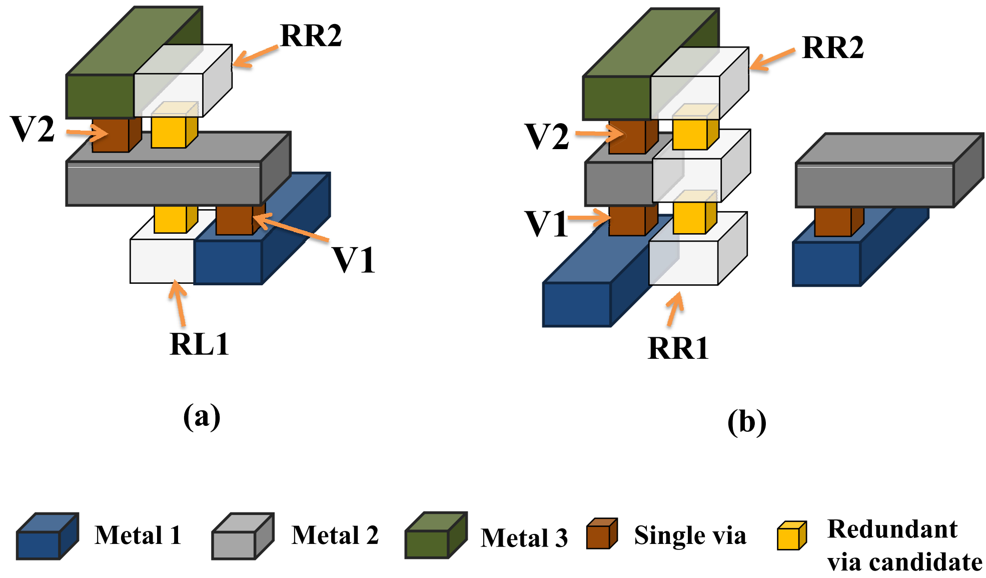


Fig. 3.3: Two feasible cases without violating the design rule.

choosing r_1 and r_2. Moreover, if r_1 and r_2 originate from the same single via, there wouldn't exist an edge $e(r_1, r_2)$ to connect r_1 and r_2. ■

Remark of Definition 2: The E_c wouldn't contain an edge $e(r_1, r_2)$ to connect r_1 and r_2 if they belong to the same net and wouldn't violate any design rule for simultaneously choosing them. Two examples are shown in Fig. 3.3. Both $RL1$ and $RR2$ in Fig. 3.3(a) can simultaneously exist, and both $RR1$ and $RR2$ in Fig. 3.3(b) can simultaneously exist because they do not violate any design rule.

Definition 3 (Mixed bipartite-conflict graph/MBC graph) *A mixed bipartite-conflict graph $G = (V \cup R, E_v \cup E_c)$ is a union of $G_v = (V \cup R, E_v)$ and $G_c = (R, E_c)$. Here, the vertex set of G is equal to $V \cup R$, and the edge set of G is equal to $E_v \cup E_c$.* ■

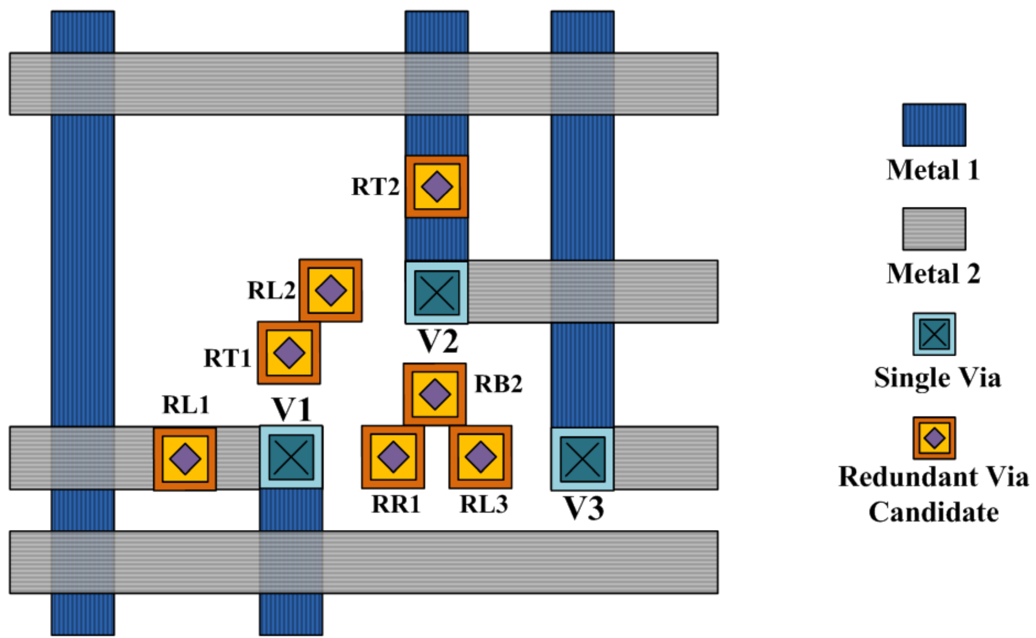
Fig. 3.4 gives an example to illustrate the above graphs. Fig. 3.4(a) shows a portion of routed circuit which has three single vias and seven redundant via candidates, and it shows the conflicts of their redundant via candidates. Fig. 3.4(b) illustrates the corresponding via-candidate bipartite graph of this routed circuit. The edge set E_v is composed of all edges in

Fig. 3.4(b). Fig. 3.4(c) is the corresponding candidate relative graph of this routed circuit. The edge set E_c consists of all edges in Fig. 3.4(c). Finally, Fig. 3.4(d) is the MBC graph of this routed circuit by combining Fig. 3.4(b) and Fig. 3.4(c). The edge set E of MBC graph shown in Fig. 3.4(d) is the union of E_v and E_c.

One difference between the MBC graph and the bipartite graph in [14] is that the vertex set of MBC graph is constructed by all single vias and their redundant via candidates in all layers. The graph can represent the conflicts of redundant via candidates methodically and perform the matching effectively. Moreover, the bipartite graph formulated in [14] merges the redundant via candidate vertices if they are in conflict, treats each stack via as a unit, and the single vias which form a stack structure can only have redundant vias inserted on the same side simultaneously [18]. On the contrary, in the MBC graph, an edge is added to connect two redundant via candidate vertices if there is a conflict between them, and each single via is treated as a unit no matter it is stacked or not. For example, Fig. 3.5(a) consists two different nets with a single via ($V3$) and a stack via which consists of two single vias ($V1$ and $V2$). Fig. 3.5(b) shows the feasible redundant via candidates of these three single vias, where a vertical conflict exists between $RR1$ and $RL3$, and a horizontal conflict exists between $RR2$ and $RL3$. The $RR1$ and $RR2$ do not exist conflict because they belong to the same net and from different metal layers. Fig. 3.5(c) shows the feasible redundant via candidates of bipartite graph formulation in [14], where $RR1$ and $RL2$ are merged to one candidate because $V1$ and $V2$ are stacked. Note that some available redundant via candidates are missed such as $RB1$ and $RL2$. Fig. 3.5(e) gives the bipartie graph of Fig. 3.5(c), where vertex $RR1$ and vertex $RR2$ are merged into one vertex $RR12$, and vertices $RR12$ and $RL3$ are also merged into a vertex R because they are in conflict. Fig. 3.5(d) gives the MBC graph of Fig. 3.5(b). Instead of merging the conflict vertices $RR1$ and $RL3$, and $RR2$ and $RL3$, we add an edge between $RR1$ and $RL3$, and an edge between $RR2$ and $RL3$.

The post-routing redundant via insertion problem can be reformulated as a mixed bipartite-conflict graph matching problem which is described below.

Mixed bipartite-conflict (MBC) graph matching problem: *Given a MBC graph $G = (V \cup R, E_v \cup E_c)$, the problem is to find a matching M of the given MBC graph. Here, M is a set of*



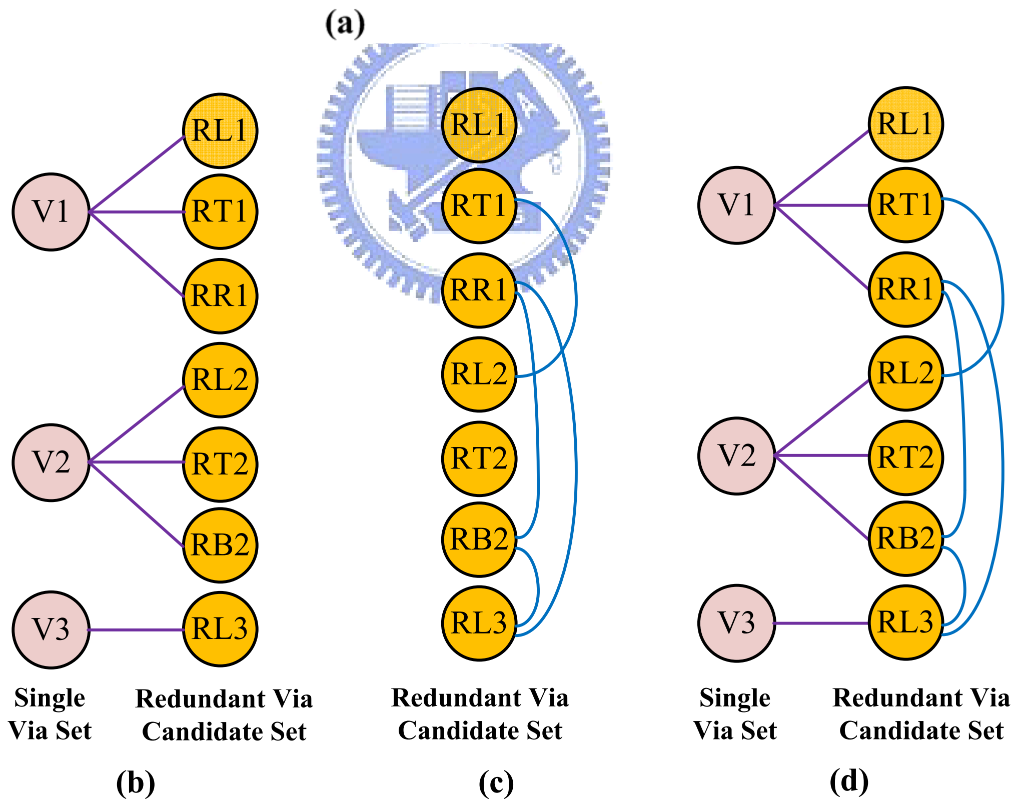


Fig. 3.4: An example of mixed bipartite-conflict graph. (a) A portion of routed circuit. (b) The via-candidate bipartite graph. (c) The candidate relative graph. (d) The mixed bipartite-conflict graph.

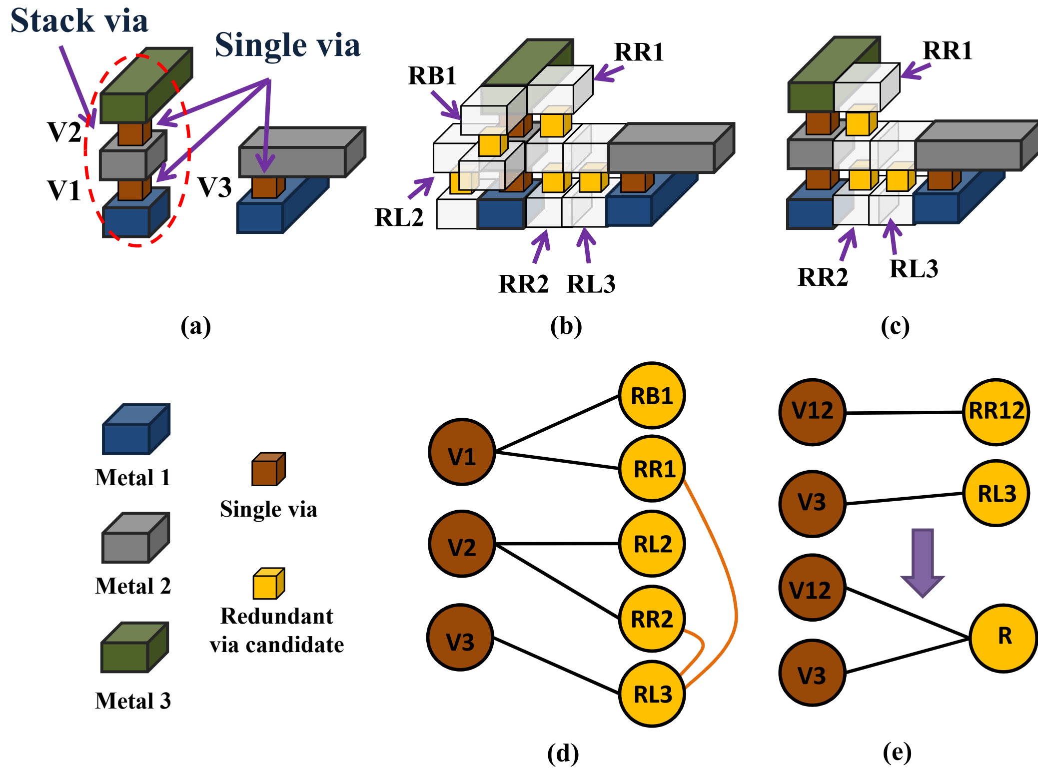


Fig. 3.5: Illustration of the difference between the mixed bipartite-conflict graph and the bipartite graph

pairwise non-adjacent edges, and the endpoints of each edge consist of one in V and the other in R. Furthermore, for two arbitrary endpoints $r_1 \in R$ and $r_2 \in R$, it is not allowed to exist an edge between them in E_c. ■

3.4 Redundant Via Insertion Method with Wire Pushing Capability (RVI^{f}A-WP)

The executing flow of the proposed RVI^{f}A-WP method is summarized in Fig. 3.6. Although simultaneously dealing with alive vias and dead vias can achieve a better insertion rate to all single vias, the RVI^{f}A-WP method inserts the redundant vias of alive vias firstly and then insert the redundant vias of dead vias for not altering the original routed design too much. Given a cell library with the design rules (such as Library Exchange Format file–LEF) and a post routed circuit design (such as Design Exchange Format file–DEF), firstly, a MBC graph is built for those alive vias, and the edge weight assignment procedure is performed for this MBC graph.

Then, according to the weights of edges, we utilize a developed heuristic minimum weight matching (HMWM) algorithm to solve the MBC graph matching problem, insert redundant vias adjacent to alive vias and update the resources of the original layout design. After the insertion for alive vias, dead vias can be protected as well by using the wire pushing capability. Similarly, a weighted MBC graph can be built for dead vias, and our HMWM algorithm can also be employed for this graph. After executing the HMWM algorithm, the inserted solution of dead vias can be obtained, the resources are update and the metal wire segments are shifted. Lastly, we output a modified layout design. In the following, we detail the proposed (RVI^{f}A-WP) method.

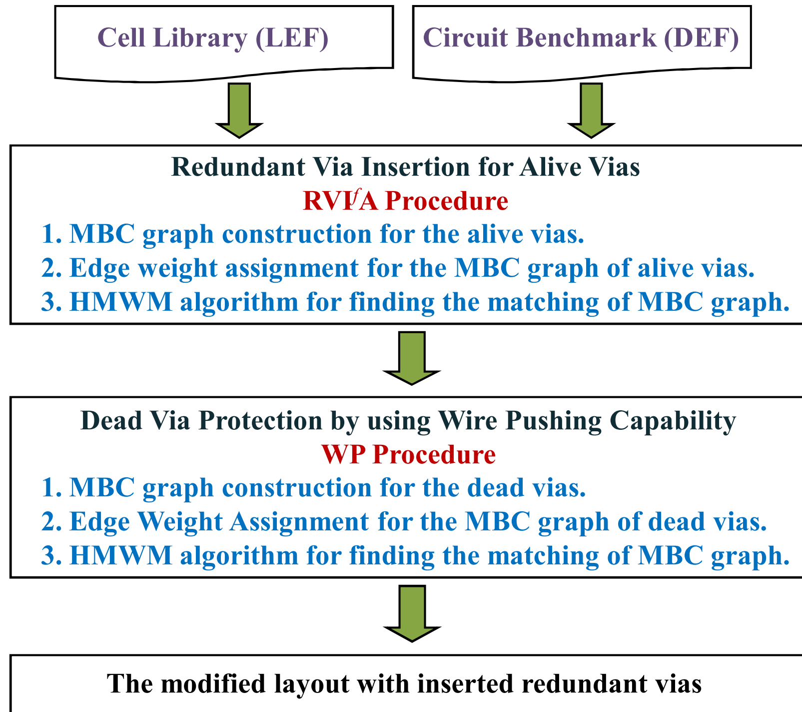


Fig. 3.6: The flowchart of the proposed RVI^{f}A-WP method.

3.4.1 Redundant Via Insertion for Alive Vias (RVI^{f}A)

In this subsection, we present the procedure of redundant via insertion for alive vias in our proposed RVI^{f}A-WP method. In the following, we name it "RVI^{f}A procedure".

MBC Graph Construction

We refer to the similar concept of graph construction algorithm (GCA) proposed in [9] to construct the corresponding MBC graph $G_A = (V \cup R, E_v \cup E_c)$ for alive vias of the given routed circuit; it utilizes the box DVE and DRW defined in [9] to find redundant via candidates of single vias, and the conflicts between the redundant via candidates of different single vias. The MBC graph can be obtained by modifying the conflict graph proposed in [9]. Our proposed MBC graph can be obtained by adding the single via vertices and constructing an edge which connects single via vertex and each of its candidate vertices to the conflict graph, and removing all the edges of candidates which belong to the same single via in the conflict graph.

Edge Weight Assignment for the MBC Graph

After constructing a MBC graph $G_A = (V \cup R, E_v \cup E_c)$ for alive vias, a value of weight $w(e)$ for each edge $e(v, r) \in E_v$, where $e \in E_v$, $v \in V$, and $r \in R$, is assigned according to the properties of the alive via vertex v and its redundant via candidate vertex r. The edge weight $w(e)$ is given as

$$w(e) = \alpha_A \times F.N. + \beta_A \times C.D. + \gamma_A \times C.T. + \sigma_A \times A.D.C., \tag{3.1}$$

Here, α_A, β_A, γ_A and σ_A are user-specified constants and the value of $w(e)$ is according to several keys which are defined as follows.

- Feasible number ($F.N.$): As defined in [9], it is the number of the feasible redundant via candidate vertices of the alive via vertex v.

- Conflict degree ($C.D.$): It is equal to the number of conflicts between redundant via candidate vertex r and other redundant via candidate vertices of different single vias causing design rule violation, i.e., the number of connected edges of the redundant via candidate vertex r in E_c .

- Candidate type ($C.T.$): The on-track redundant via is more critical than the off-track redundant via due to the better electrical properties of the on-track redundant via. Thus, we prefer to insert the on-track redundant via. The value of $C.T.$ is equal to 1 if the redundant via candidate vertex r is off-track; otherwise, it is 0 for the on-track redundant via candidate vertex.

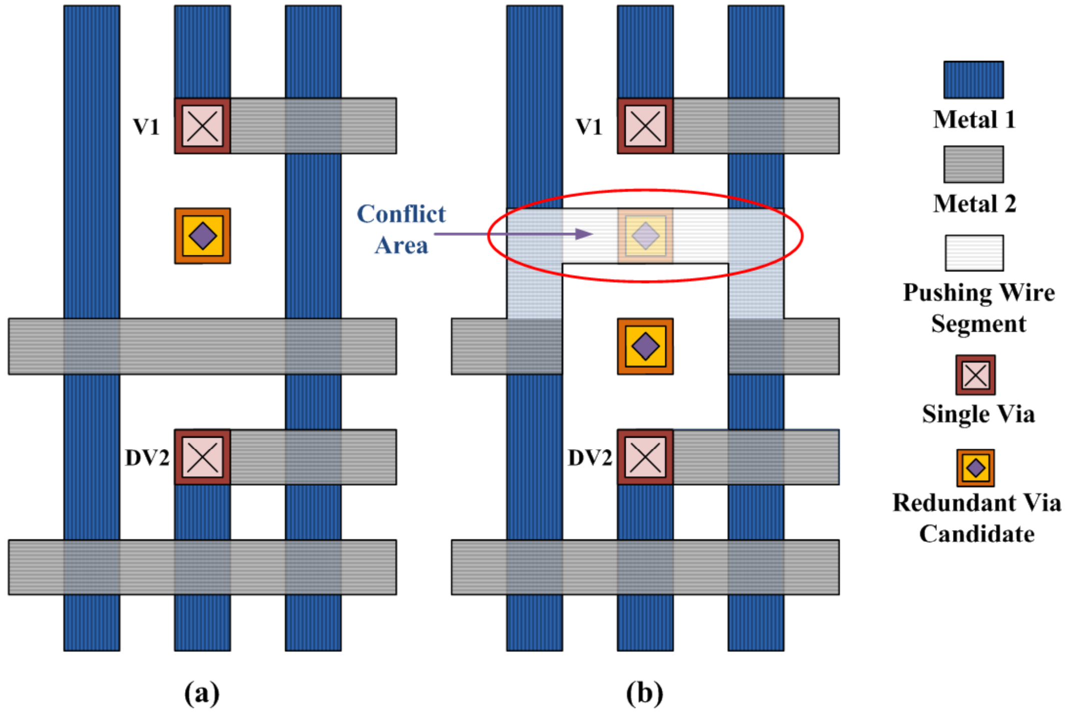


Fig. 3.7: Illustion of the conflict between two redundant via candidates for the alive and dead vias.

- Alive-dead conflict degree ($A.D.C.$): It is defined as the number of the conflicts between redundant via candidate vertex r and the redundant via candidate vertices of dead vias by pushing wire segments. Fig. 3.7 illustrates this definition. The example shown in Fig. 3.7(a) has a dead via (DV2) and an alive via (V1), and each has a redundant via candidate. In Fig. 3.7(b), the white metals are pushing wire segment, and the red circle indicates a conflict between the pushing wire segment and the redundant via candidate of the alive via.

To make a trade-off between the redundant via insertion rate and the on-track redundant via insertion rate, the order of importance is the feasible number,the candidate type, the conflict degree and the alive-dead conflict degree; hence we have $\alpha_A \geq \gamma_A \geq \beta_A \geq \sigma_A$. According to the edge weight assignment, we give the higher priority to an edge which has a smaller feasible number of its corresponding alive via vertex, and a smaller conflict degree, a smaller alive-dead conflict degree and the on-track structure of its corresponding redundant via candidate vertex. The intention of using $A.D.C.$ is to retain empty space for pushing wire segments to insert the

```
HMWM(G = (V ∪ R, E_v ∪ E_c))
Sort the edges of the edge set E_v in the non-decreasing order weight
 by constructing the binary search tree– BST.
while existing edges in E_v can be added to the matching M
  Pick an e(v, r) ∈ E_v, where v ∈ V and r ∈ R, which has the
   smallest weight in the binary search tree–BST.
  Add e(v, r) to M,
  Do Update(e(v, r), G),
return matching M

Update(e(v, r), G)
1. Update the vertex set V ∪ R and edge set E_v ∪ E_c of G:
   a) Delete all adjacent vertices of r except v and their connected
      edges, and the candidate vertices of v except r and their
      connected edges.
   b) Delete the edge e(v, r) and vertices v and r.
2. Update the edge weights of G:
   Re-calculate the value of conflict degree and the feasible number of
   each edge e(v_1, r_1) ∈ E_v with v_1 ∈ V and r_1 ∈ R. Here,
   at least one of the redundant via candidate vertices of v_1 is deleted,
   or r_1 connects to deleted redundant via candidate vertices.
3. Update the binary search tree–BST.
```

Fig. 3.8: Algorithm of HMWM.

redundant via adjacent to the dead via.

HMWM Algorithm for the MBC Graph

The HMWM algorithm is a heuristic method for solving the MBC graph matching problem. After constructing the MBC graph $G_A = (V \cup R, E_v \cup E_c)$ of alive vias, where each edge $e \in E_v$ has a specific weight, the RVI^{f}A procedure performs the HMWM algorithm to solve the weighted MBC graph for obtaining the matching M, then it inserts a redundant via next to each matched alive via. The HMWM algorithm firstly sorts the edges which are in the edge set E_v according to their weights in the non-decreasing order by using the binary search tree (BST), then it adds an edge having the smallest weight to the matching M. If the number of the edges with the smallest weight is more than one, it randomly select one and add it to M. After one edge has been added, the MBC graph is modified immediately by deleting all adjacent vertices of the matched redundant via candidate vertex and its connected edges. The candidates of the matched single via vertex and its connected edges are also removed. Then, the conflict degree

and the feasible number of each relative edge $e(v_1, r_1) \in E_v$ are re-calculated. Here, v_1 is a single via vertex, and at least one of its candidate vertices is deleted or r_1 connects to deleted redundant via candidate vertices. In addition, the binary search tree is also updated. The above process is terminated until no edge can be added to M, and the matching of the weighted MBC graph is obtained. The computational procedure is shown in Fig. 3.8.

Procedure Summarization of Redundant Via Insertion for Alive Vias

Given a routed circuit, the steps of RVI^{f}A procedure are summarized as follows.

Step 1: Check the surrounding environment to find redundant via candidate vertices and conflicts for all single via vertices, and add dead via vertices to the dead via set (DVSET).

Step 2: Check the surrounding environment to find redundant via candidate vertices and conflicts for all dead via vertices and determine how many wire segments need to be pushed.

Step 3: Construct a MBC graph $G_A = (V \cup R, E_v \cup E_c)$ for all alive vias, and assign a weight to each edge $e \in E_v$ according to those keys mentioned in section 3.4.1.

Step 4: Perform HMWM algorithm to find the matching of this weighted MBC graph. Then, unmatched via vertices become dead via vertices and are added to the dead via set (DVSET).

Step 5: Modify the layout resources which include redundant vias and extra metal wires. The redundant via candidate vertices of dead vias which are conflicted with the redundant via vertices of matched alive via vertices are removed.

An Example of the RVI^{f}A Procedure

Fig. 3.9 gives an example to illustrate the RVI^{f}A procedure. Fig. 3.9(a) presents a routed circuit with its redundant via candidates and potential pushing wire segments. $V1$, $V2$ and $V3$ are alive vias and their redundant via candidates are $\{RL1, RT1, RR1\}$, $\{RL2, RT2, RB2, RR2\}$ and $\{RL3\}$, respectively. $V4$ is a dead via and has a redundant via candidate $RB4$ by using wire pushing capability. Note that the candidate $RB4$ of dead via $V4$ and the candidate $RT2$ of alive via $V2$ cannot both exist in the design because they cause a conflict happened and violate the design rule.

In Step3, we construct a MBC graph $G_A = (V \cup R, E_v \cup E_c)$ for alive vias in Fig. 3.9(a) and assign a weight to each edge as shown in Fig. 3.9(b). Here $\alpha_A = 3$, $\beta_A = 1$, $\gamma_A = 2$ and

$\sigma_A = 0.1$. We give an example to explain the edge weight assignment. For edge $e(V3, RL3)$, vertex $V3$ has an off-track redundant via candidate vertex $RL3$. $RL3$ conflicts with $RB2$ and $RR1$ but it does not have any conflict with redundant via candidates of dead vias. Thus the weight of e is $w(e) = 3 \times 1 + 1 \times 2 + 2 \times 1 + 0.1 \times 0 = 7$. The process of heuristic minimum weight matching (HMWM) in Step4 is shown in Fig. 3.9(b) to Fig. 3.9(e). In Fig. 3.9(b), the edge $e(V3, RL3)$ which has the smallest weight 7 is first extracted to the matching, then we delete the adjacent vertices $\{RR1, RB2\}$ of RL3 and their connected edges $\{e(V1, RR1), e(RR1, RB2), e(RR1, RL3), e(V2, RB2), e(RB2, RL3)\}$. After the deletion, the weights of the edges $\{e(V1, RL1), e(V1, RT1), e(V2, RL2), e(V2, RT2), e(V2, RR2)\}$ need to be updated. The process is continuously executed to extract edges of the weighted MBC graph in the non-decreasing order weight, until no edge can be added. After match one edge, we need to delete the relative vertices and their connected edges, and update the weight of the relative edges. Fig. 3.9(e) shows the matching $\{(V1, RL1), (V2, RR2), (V3, RL3)\}$ of the given routed circuit. After Step 5, the modified layout is shown in Fig. 3.9(f).

3.4.2 Dead Via Protection by using Wire Pushing Capability (WP)

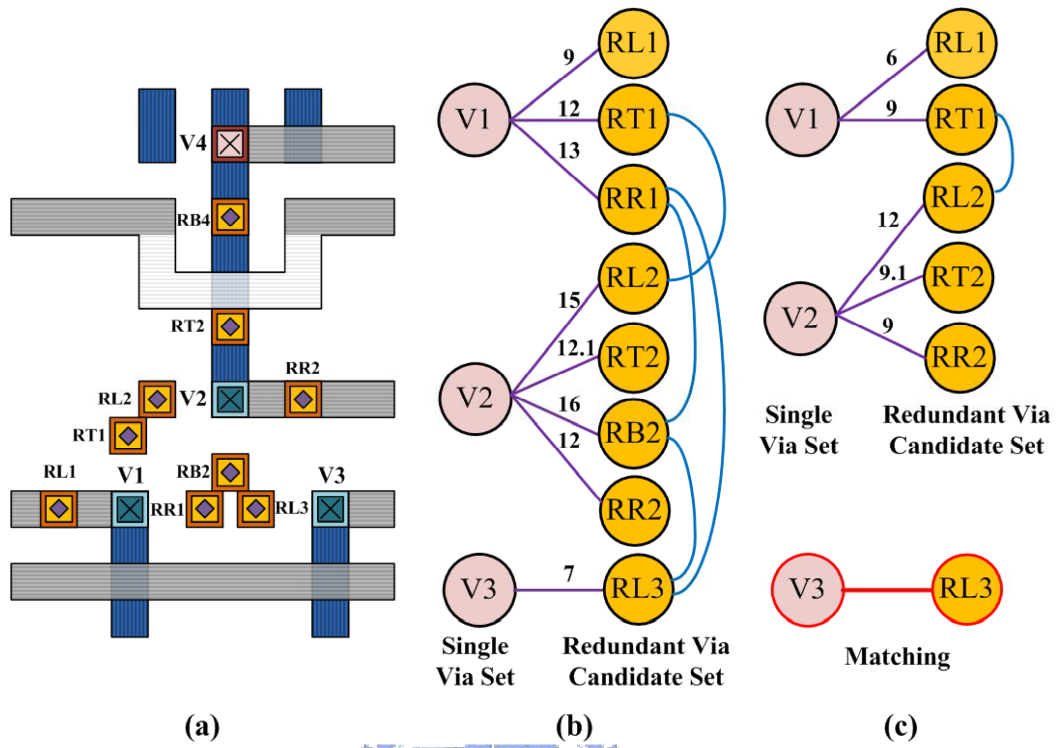
After redundant vias have been inserted for alive vias, redundant vias are inserted next to dead vias by using the wire pushing capability to further enhance the yield of vias. In the following, this procedure which we name it “WP procedure” is detailed.

MBC Graph Construction

In the DVSET, there are two types of dead vias. One is the original dead vias of the routed circuit, and the other is the new dead vias which are those unmatched alive vias in the RVI^f A procedure. For the DVSET, we construct a MBC graph $G_D = (V \cup R, E_v \cup E_c)$ for dead vias as the construction of the MBC graph $G_A = (V \cup R, E_v \cup E_c)$ for alive vias of the routed circuit. We utilize the searching region (see 3.2) as the DRW defined in [9] to find redundant via candidates and conflicts of candidates for dead vias.

Edge Weight Assignment for the MBC Graph

After constructing a MBC graph $G_D = (V \cup R, E_v \cup E_c)$ for dead vias, a weight $w(e)$ for each edge $e(v, r) \in E_v$, where $v \in V$ and $r \in R$, is assigned according to the properties of



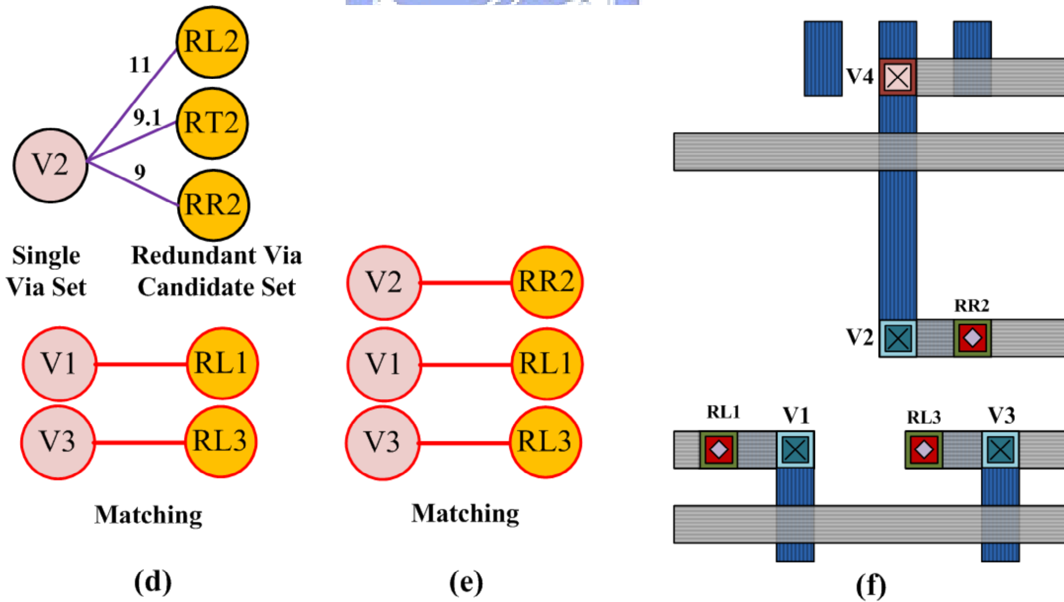


Fig. 3.9: An example for illustrating the RVI^{f}A procedure.

dead via vertex v and its redundant via candidate vertex r. The edge weight $w(e)$ is given as

$$w(e) = \alpha_D \times D.F.N. + \beta_D \times D.C.D. + \gamma_D \times P.W.N. + \sigma_D \times C.T., \quad (3.2)$$

where α_D, β_D, γ_D and σ_D are user-specified constants, and the value of $w(e)$ is according to several keys defined as follows.

- Dead-Feasible number ($D.F.N.$): It is the number of the feasible redundant via candidate vertices of dead via vertex v by using wire pushing capability.
- Dead-Conflict degree ($D.C.D.$): The dead-conflict degree of redundant via candidate vertex r of dead via vertex v is equal to the number of conflicts between r and the other redundant via candidate vertices of dead vias by pushing wire segments. An example is shown in Fig. 3.10. There is an intersection between two pushing wire segments of different dead via's candidates $\{DR1, DL2\}$.
- Pushing wire number ($P.W.N.$): It is the number of pushing wire segments for allocating enough space to insert r. For example, the $P.W.N.$ of $DR1$ is two and the $P.W.N.$s of $DL2$ and $DR2$ are one as shown in Fig. 3.10. We intends to select a redundant via candidate vertex whose $P.W.N.$ is smaller since the movement of wires is less.
- Candidate type ($C.T.$): It is equal to 1 if the redundant via candidate vertex r is off-track, otherwise it is 0 for on-track redundant via candidate vertex r.

The order of importance is the dead-feasible number, the dead-conflict degree, the pushing wire number and the candidate type; hence we have $\alpha_D \geq \beta_D \geq \gamma_D \geq \sigma_D$ in the edge assigment. According to the edge weight assignment, we also give a higher priority to an edge having a smaller dead-feasible number of its corresponding dead via vertex, and a smaller dead-conflict degree, a smaller pushing wire number and the on-track structure of its corresponding redundant via candidate vertex.

HMWM Algorithm for the MBC Graph

After constructing the MBC graph $G_D = (V \cup R, E_v \cup E_c)$ and assigning a suitable weight of each edge $e \in E_V$, we also apply HMWM algorithm to find the matching of this weighted

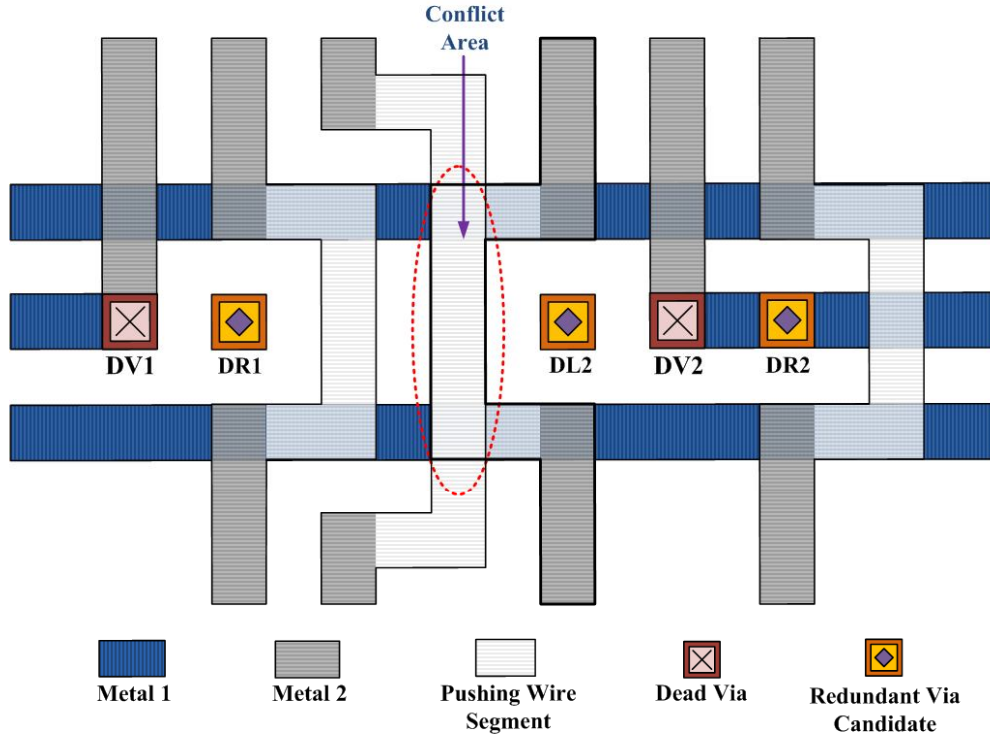


Fig. 3.10: Illusion of a conflict between two pushing wire segments

MBC graph. The detailed process of HMWM algorithm to solve $G_D = (V \cup R, E_v \cup E_c)$ is the same as presented in 3.4.1.

Procedure Summarization of Dead Via Protection by using Wire Pushing Capability

Given a routed circuit, the detailed steps of WP procedure for solving MBC graph $G_D = (V \cup R, E_v \cup E_c)$ of dead vias are shown as follows.

Step 1: Check the surrounding environment to find redundant via candidate vertices and conflicts for new dead via vertices and determine how many wire segments need to be pushed.

Step 2: Construct a MBC graph $G_D = (V \cup R, E_v \cup E_c)$ for all dead vias, and assign a weight to each edge $e \in E_v$ according to those keys mentioned in section 3.4.2.

Step 3: Perform HMWM algorithm to find the matching of this weighted MBC graph.

Step 4: Shift the pushing wire segments and modify the layout resources including the redundant vias and extra metal wires.

An Example of the WP Procedure

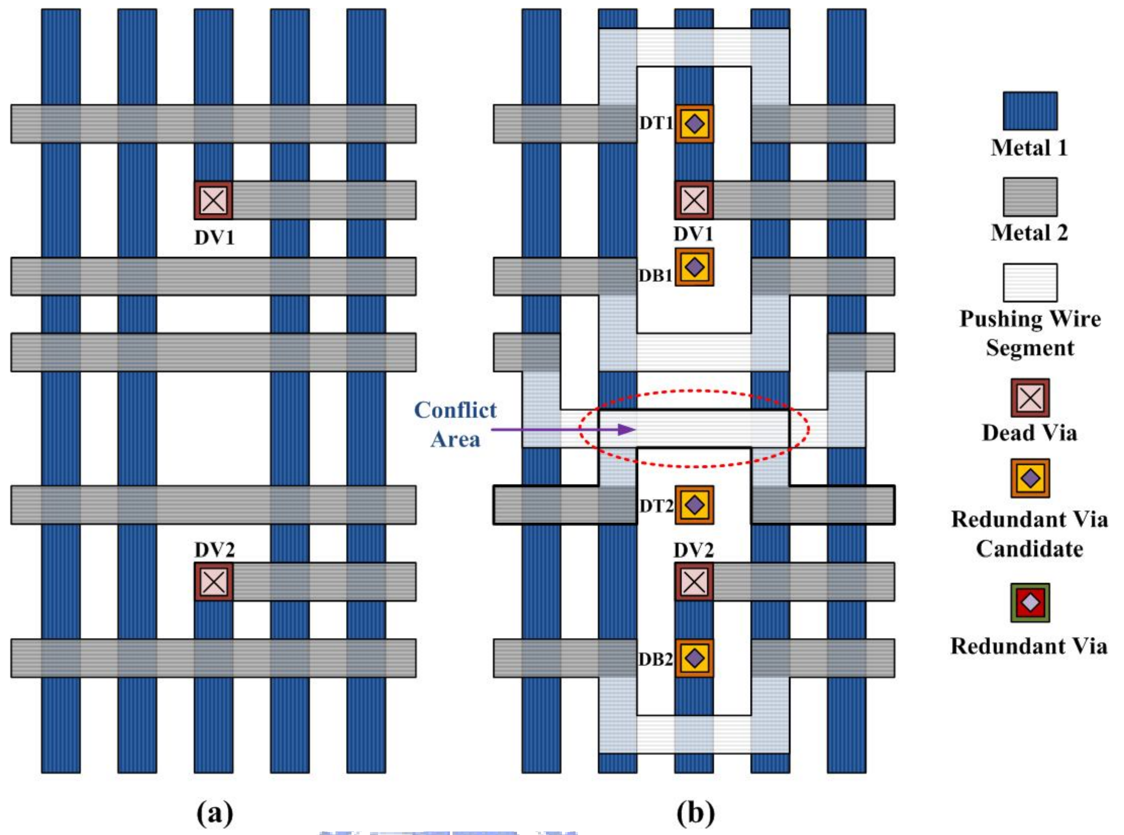
Fig. 3.11 gives an example to illustrate the WP procedure. Fig. 3.11(a) is a a portion of routed circuit. In Fig. 3.11(b), $DV1$, $DV2$ are dead vias with their redundant via candidates $\{DT1, DB1\}$, $\{DT2, DB2\}$ by using wire pushing capability, and there is a conflict between two pushing wire segments of $DB1$ and $DT2$. In Step2, we construct a MBC graph $G_D = (V \cup R, E_v \cup E_c)$ for dead vias and assign a weight to each edge in E_v is shown in Fig. 3.11(c). Here $\alpha_D = 3$, $\beta_D = 1$, $\gamma_D = 1$ and $\sigma_D = 0.1$. We also give an example to explain the edge weight assignment. For edge $e(DV2, DT2)$, the vertex DV2 has an off-track redundant via candidate vertex $DT2$, the pushing wire segments of $DT2$ and $DB1$ are in conflict, and the number of pushing wire segments of $DT2$ is one. Thus the weight of e is $w(e) = 3 \times 2 + 1 \times 1 + 1 \times 1 + 0.1 \times 1 = 8.1$. The process of HMWM algorithm in Step 3 is shown in Fig. 3.11(c) to Fig. 3.11(e). In Fig. 3.11(c), both edges $e(DV2, DB2)$ and $(DV1, DT1)$ have the smallest weight 7. We randomly select an edge which is $e(DV2, DB2)$ and add it to the matching. After that, we delete the redundant via candidate vertex $DT2$ of matched vertex $DV2$ and its connected edges $e(DV2, DT2)$ and $e(DB1, DT2)$, and updates the weights of $e(DV1, DT1)$ and $e(DV1, DB1)$. Finally, the matching$\{(DV1, DT1), (DV2, DB2)\}$ is obtained as shown in Fig. 3.11(e). After Step 4, the modified layout is shown in Fig. 3.11(f).

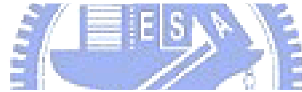
3.4.3 Runtime Complexity Analysis of HMWM Algorithm

Given a weighted MBC graph $G_A = (V \cup R, E_v \cup E_c)$ constructed from a practical routed circuit for alive vias, firstly, the HMWM algorithm presented in Fig. 3.8 sorts the edge set E_v in $O(|E_v| \log |E_v|)$ time.

The HMWM algorithm takes constant time to pick an edge from the sorted edge set E_v and add this picked edge to the matching set for each iteration.

Since the number of conflicts for each redundant via candidate in a practical layout geometry can be bounded by a fixed number k_1, the number of conflicts for each redundant via candidate vertex in R is also bounded by k_1. The feasible number (F.N.) of each alive via vertex is bounded by 4. Therefore, the number of vertices need to be deleted is bounded by $k_1 + 4$. Since the number of edges induced by each redundant via candidate vertex is bounded by $k_1 + 1$, the number of edges which need to be deleted is bounded by $(k_1+4)(k_1+1)$. Because each deletion takes constant time, the deletion step can be performed in $O(k_1^2)$ time for each iteration.





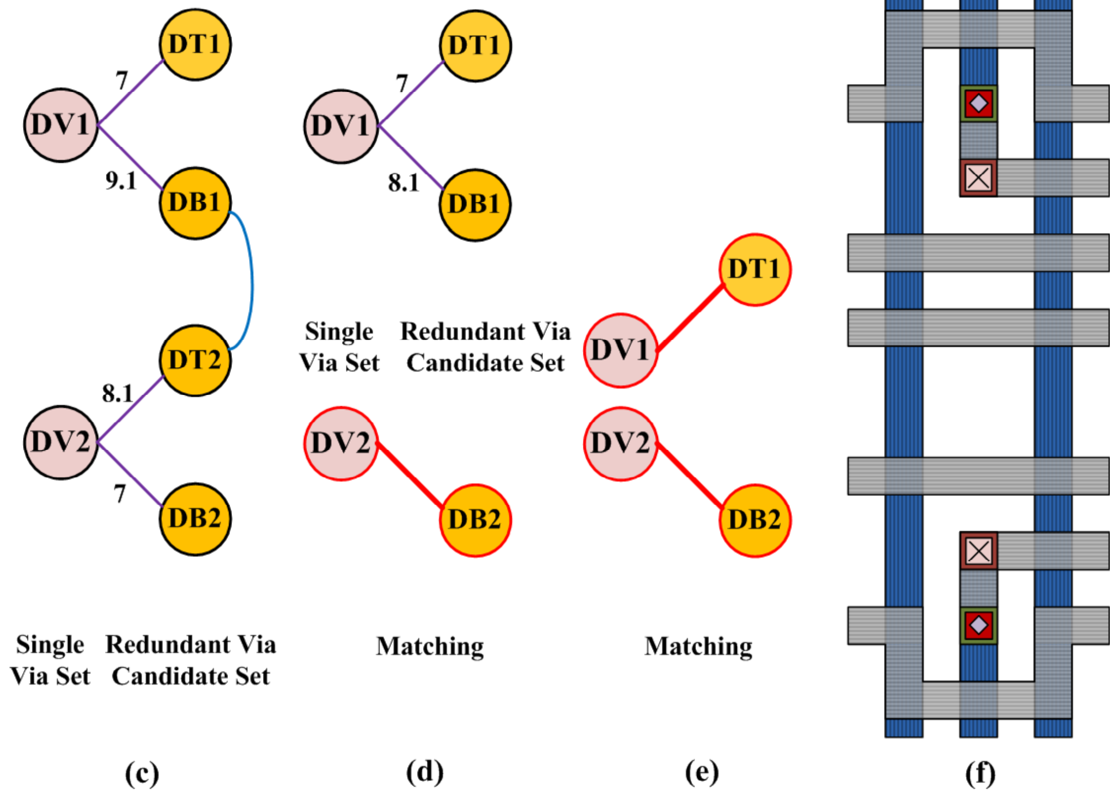


Fig. 3.11: An example for illustrating the WP procedure.

Similarly, the number of edges which need to be updated can be analyzed to be bounded by a constant $k_1^2 + 5k_1$, and each takes constant time. After updating their values, each one takes $O(\log |E_v|)$ time to update the binary tree–BST. Hence, the runtime complexity of the updating procedure is $O(k_1^2 \log |E_v|)$ time for each iteration.

From the avove description, the runtime complexity of each matching iteration is $O(k \log |E_v|)$. Here, $k = k_1^2$.

Because the number of matching iterations is bounded by $|E_v|$, the complexity of the while loop shown in Fig. 3.8 is $O(k|E_v| \log |E_v|)$. Finally, we conclude that the executing time of HMWM algorithm for solving a weighted MBC graph is in $O(|E_v| \log |E_v|)$ time.

According to [21, 22], the maximum bipartite matching can solve a bipartite graph in $O(\sqrt{|V \cup R|}|E_v|)$ time, and the minimum weighted bipartite matching can solve a weighted bipartite graph in $O(|V \cup R|^2 \log |V \cup R| + |V \cup R||E_v|)$ time. Obviously, the HMWM algorithm is faster than both them.

3.4.4 Speed Up

As the experiments shown in [18, 19], all of the corresponding conflict graphs of the circuits they used are sparse. Thus, they compute the connected components of a conflict graph to accelerate for solving their problem. From the citcuits used in our experiments, the corresponding MBC graphs are also sparse in general case. Therefore, we can also group the connected components of MBC graph $G = (V \cup R, E_v \cup E_c)$ into many subgraphs by using depth first search algorithm [23], and perform the HMWM algorithm for each subgraph. Note that the each subgraph is also MBC graph. This method can accelerate our approach without losing any insertion rate, and this accelerated method can also be utilized in maximum bipartite matching algorithm.

Chapter 4

Experimental Results

The RVI^{f}A-WP method has been implemented in C++ programming language on a dual core 2.13-GHz PC machine with 4-GB memory. Firstly, the RVI^{f}A procedure of RVI^{f}A-WP method is compared with the TDVI algorithm [14] for the post-layout redundant via insertion problem of alive vias. The proposed HMWM algorithm was employed to solve the MBC graph, and TDVI was utilized to solve the bipartite graph. The benchmarks were generated by the authors of [14]. "Mcc1" and "Mcc2" contain 4 metal layers and other benchmarks contain only three metal layers. The connected components of the corresponding graphs were computed, and the corresponding graphs were divided into many subgraphs to accelerate the matching procedures for both HMWM and TDVI algorithms. We chose $\alpha_A = 3$, $\beta_A = 1$, $\gamma_A = 2$ and $\sigma_A = 0.1$ for the HMWM algorithm to achieve a better trade-off between the redundant via insertion rate and the on-track via insertion rate. The LEDA package [22] was used in TDVI for solving the maximum bipartite matching and minimum weighted bipartite matching problems.

The results of RVI^{f}A procedure and TDVI algorithm are shown in Table 4.1. In the table, TDVI has two modes, the insertion rate mode and the on-tract/stack redundant via enhancement mode. A stack via and its redundant via are counted as two single vias and two redundant vias in the result of TDVI, respectively. The "#Single Via" is the total single vias of the given layout, "#Alive Via" is the number of alive vias, "#Ins. RVia" is the number of redundant vias after performing the insertion method, "Ins. Rate" and "On-T. Rate" are the redundant via insertion rate and on-track redundant via insertion rate, respectively, "Time of HMWM" reports the runtime of HMWM algorithm, and "Time" reports the runtime of TDVI.

From Table 4.1, it can be observed that our approach can obtain average 11.24$\times$ runtime

speed up over the TDVI algorithm and achieve an average insertion rate at 99.54%. Both the insertion rate and the on-track insertion rate are higher than the TDVI[1] algorithm for each benchmark circuit. Moreover, the RVI^{f}A procedure can achieve a more better insertion rate and a more better on-track insertion rate in the circuits with more than three metal layers, such as "Mcc1" and "Mcc2". It is because that TDVI needs to partition the given layout into sub-layouts and solves them in these cases. This heuristic process might degrade the quality of insertion solution. However, the proposed RVI^{f}A procedure simultaneously considers all single vias in all layers of the circuit in the matching process. Beside, It should be mentioned that we can assign other values to α_A, β_A, and γ_A according to the significance of insertion rate and on-track rate. Therefore, the insertion solution can get a higher insertion rate with little loss of on-track rate or opposite result.

After executing the RVI^{f}A procedure for alive vias on each benchmark, the WP procedure can be performed to protect dead vias, and the result is presented in Table 4.2. The maximum wire shifted degree for the searching region was set to be 3 for each metal layer, and we chose $\alpha_D = 3$, $\beta_D = 1$, $\gamma_D = 1$ and $\sigma_D = 0.1$ in the HMWM algorithm for solving the MBC graph of dead vias. The "#Dead Via" and "#Extra Dead Via" are the number of original dead vias and the number of extra dead vias generated after executing the RVI^{f}A procedure, respectively. The "#All Dead Via" is the number of all dead vias and is equal to the sum of #Dead Via and #Extra Dead Via. The "#Alive Dead Via" is the number of dead vias that can be potentially protected by utilizing the wire pushing capability, "#Prot. Dead Via" gives the number of protected dead vias after pushing wire segments, "Prot. Rate" is the rate of protected dead vias, "PWLen. Percent." represents the ratio of the total length of pushing wire segments to the total length of nets, and "Time" reports the runtime of WP procedure.

Table 4.2 shows that the dead vias can be effectively protected by using the wire pushing capability. The average rate of protected dead vias can be up to 54.41% with the maximum and average percentages of pushing wire length being only 0.49% and 0.26%, respectively, it shows that the total length of pushing wire segments is trifling for the total length of nets in the

[1]In Table 4.1, it can also be found that the insertion rate of TDVI with insertion rate mode is slightly smaller than the insertion rate of TDVI with on-track/stack redundant via enhancement mode. It is because that the single via and stack via have the same priority in its insertion rate mode, hence, the single via might be matched first when they are in conflict.

circuit. Fig. 4.1 is a portion of wire pushing result of circuit S38584 after performing dead via protection by using wire pushing capability. Fig. 4.1(a) shows a portion of original routing of circuit S38584 and Fig. 4.1(b) shows the portion of wire pushing result of this circuit, we can see that there is two dead vias protected by using pushing wire capability. Fig. 4.2, Fig. 4.3 and Fig. 4.4 show the distributions of dead vias and the distributions of dead vias after wire pushing of circuit Struct, Mcc2 and S38584, respective. We enlarge the dead vias in these Figures to clearly observe the dead via distributions of the circuits , and we can see that the numbers of the dead vias has been decreased. Thus, the dead vias can be effectively protected by using wire pushing capability in the benchmarks we used. Fig. 4.5, Fig. 4.6 and Fig. 4.7 show the original routing results and the distributions of pushing wire segments of circuit Struct, Mcc2 and S38584, respective, we can see that the total length of all pushing wire segments of the circuits is quite short.

Benchmark			RVI^{f}A Procedure (Ours)				TDVI [14] with insertion rate mode				TDVI [14] with on-track/ stack redundant via enhancement mode			
Circuit Name	#Single Via	#Alive Via	#Ins. RVia	Ins. Rate (%)	On-T. Rate (%)	Time of HMWM (sec)	#Ins. RVia	Ins. Rate (%)	On-T. Rate (%)	Time (sec)	#Ins. RVia	Ins. Rate (%)	On-T. Rate (%)	Time (sec)
Mcc1	5788	5265	5216	99.07	70.90	0.031	5140	97.63	44.53	0.297	5142	97.66	69.84	0.281
Mcc2	33153	29351	29163	99.36	73.23	0.109	28757	97.98	45.43	1.547	28766	98.01	72.70	1.391
Struct	7248	7195	7194	99.99	82.71	0.031	7175	99.72	34.06	0.375	7175	99.72	79.36	0.344
Primary1	5347	5252	5252	100.00	81.23	0.016	5241	99.79	39.69	0.281	5241	99.79	79.83	0.265
Primary2	22365	21790	21783	99.97	81.89	0.094	21730	99.72	42.31	1.125	21730	99.72	80.91	1.031
S5378	6784	6341	6305	99.43	76.69	0.016	6227	98.20	47.13	0.312	6228	98.22	76.72	0.281
S9234	5350	5076	5036	99.21	80.78	0.016	4987	98.25	50.85	0.281	4988	98.27	80.61	0.250
S13207	13767	13072	12995	99.41	80.78	0.047	12885	98.57	50.39	0.656	12886	98.58	80.72	0.578
S15850	16633	15677	15595	99.48	79.44	0.047	15445	99.52	49.83	0.750	15447	98.53	79.26	0.672
S38417	40655	38577	38394	99.53	81.14	0.156	38126	98.83	50.39	1.844	38135	98.85	80.89	1.657
S38584	54483	51276	50994	99.45	79.65	0.235	50435	98.36	50.55	2.453	50446	98.38	79.62	2.219
Comparison			1	99.54	78.95	1	0.9915	98.69	45.92	12.43	0.9916	98.70	78.22	11.24

Table 4.1: Comparison for the proposed RVI^{f}A procedure with TDVI [14].

Benchmark				RVI^{f}A-WP Method								
				RVI^{f}A Procedure			WP Procedure after RVI^{f}A Procedure					
Circuit Name	#Single Via	#Alive Via	#Dead Via	#Ins. RVia	#Ins. Rate (%)	#Extra Dead Via	#All Dead Via	#Alive Dead Via	#Prot. Dead Via	Prot. Rate (%)	PWLen. Percent. (%)	Time (sec)
Mcc1	5788	5265	523	5216	99.07	49	572	463	370	64.69	0.18	0.28
Mcc2	33153	29351	3802	29163	99.36	188	3990	3268	2703	67.74	0.11	8.45
Struct	7248	7195	53	7194	99.99	1	54	47	42	77.78	0.02	0.08
Primary1	5347	5252	95	5252	100.00	0	95	84	69	72.63	0.03	0.05
Primary2	22365	21790	575	21783	99.97	7	582	504	384	65.98	0.05	0.39
S5378	6784	6341	443	6305	99.43	36	479	272	174	36.33	0.45	0.33
S9234	5350	5076	274	5036	99.21	40	314	220	137	43.63	0.49	0.30
S13207	13767	13072	695	12995	99.41	77	772	512	334	43.26	0.37	1.84
S15850	16633	15677	956	15595	99.48	82	1038	680	455	43.83	0.40	2.28
S38417	40655	38577	2078	38394	99.53	183	2261	1514	939	41.53	0.37	13.22
S38584	54483	51276	3207	50994	99.45	282	3489	2278	1434	41.10	0.42	29.31
Average	-	-	-	-	99.54	-	-	-	-	54.41	0.26	-

Table 4.2: Experimental results of the proposed RVI^{f}A-WP method.

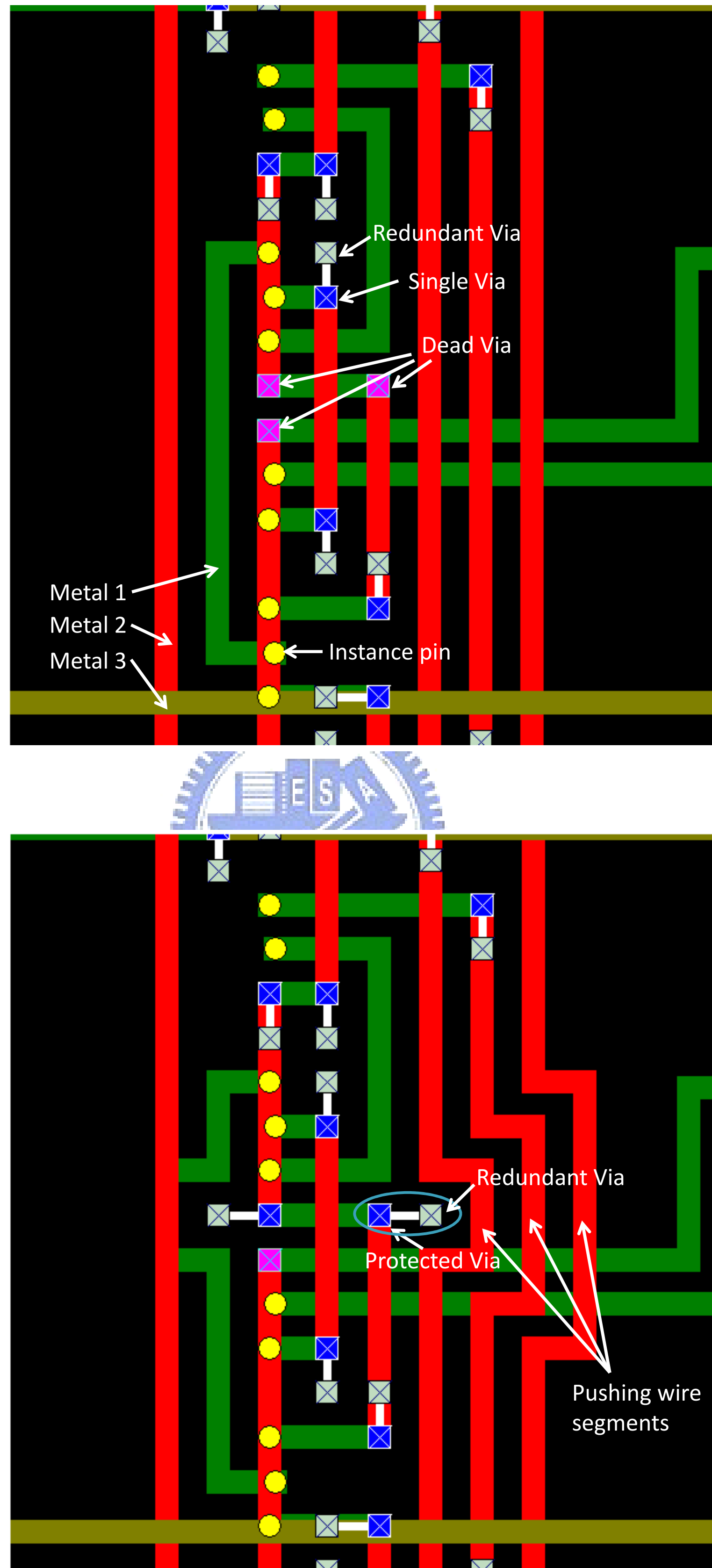
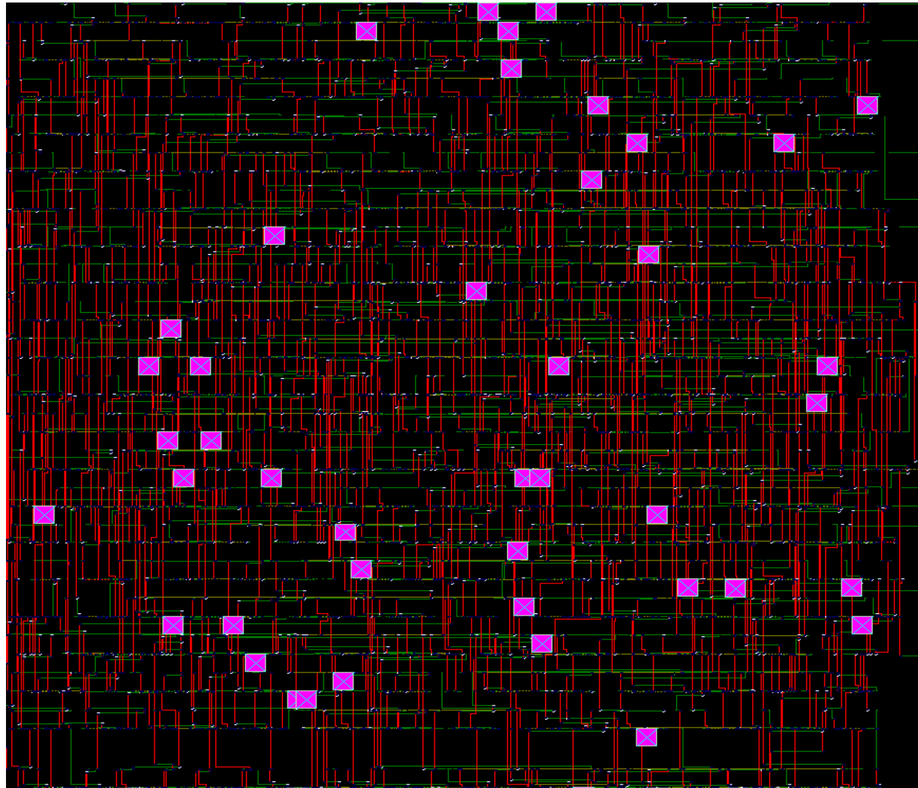


Fig. 4.1: (a) A portion of original routing of circuit S38584 (b) A portion of wire pushing result of circuit S38584



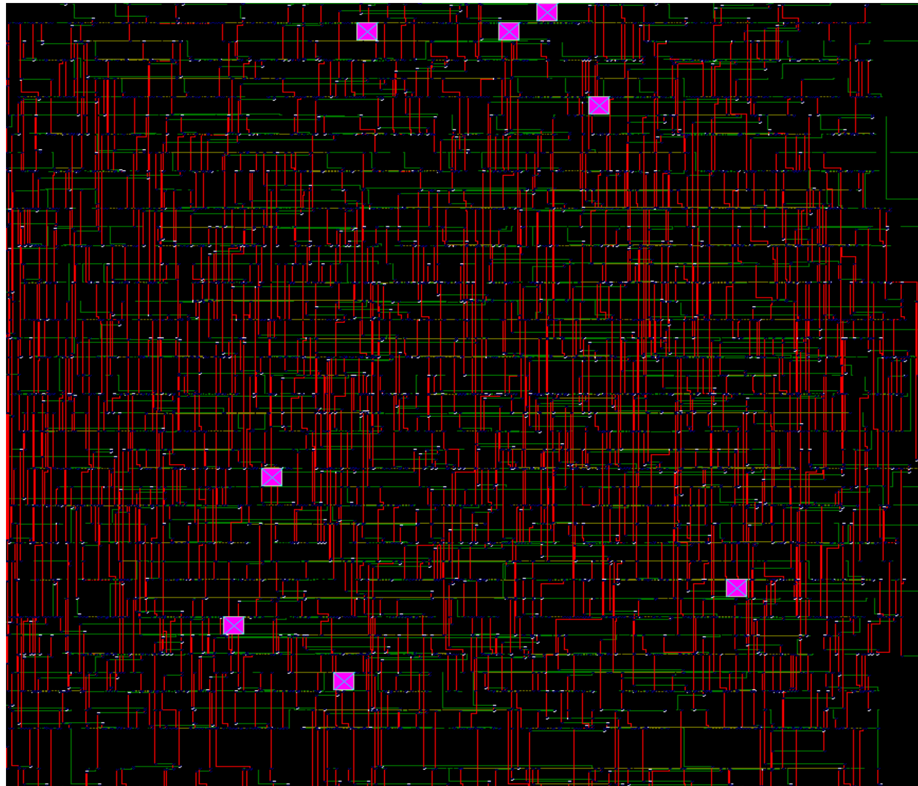


Fig. 4.2: (a) The distribution of dead vias of circuit Struct. (b) The distribution of dead vias after wire pushing of circuit Struct.

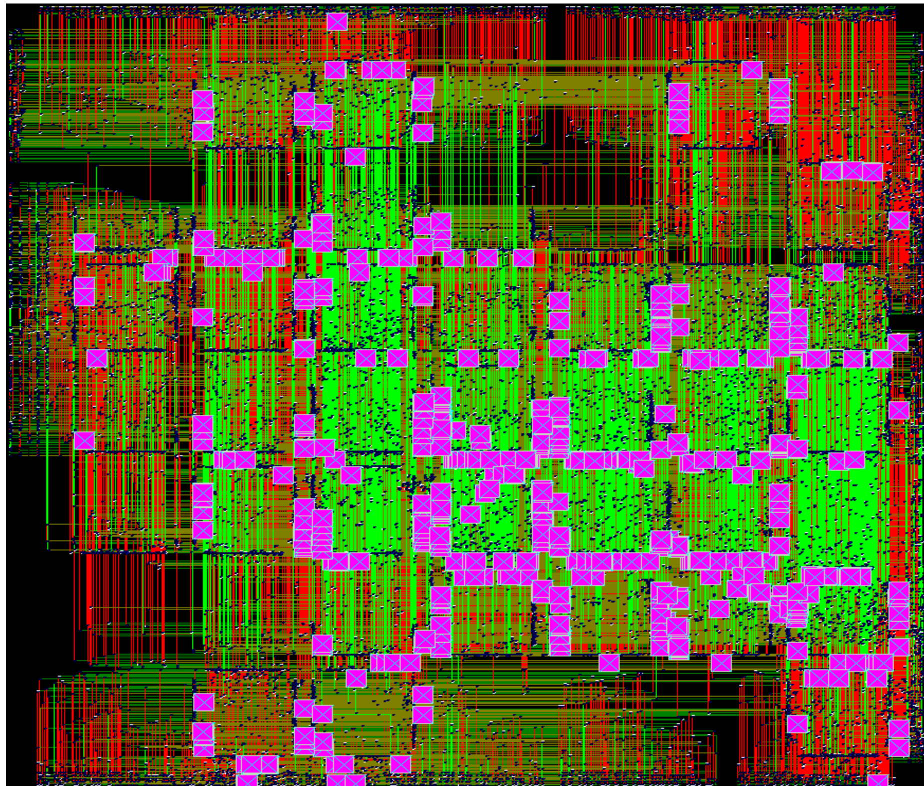
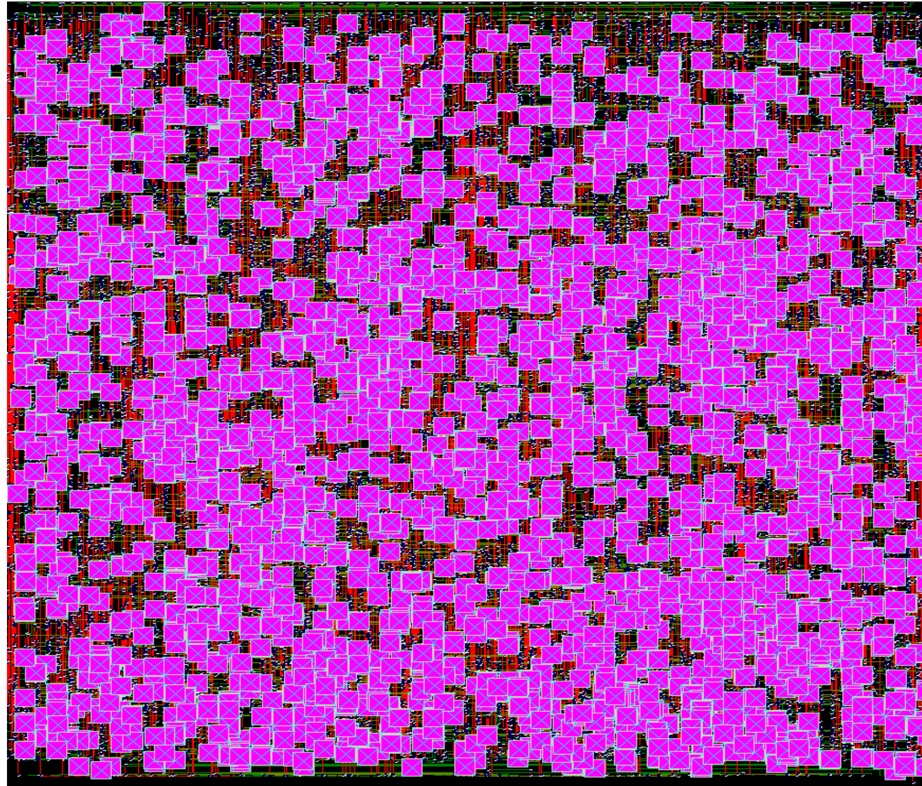


Fig. 4.3: (a) The distribution of dead vias of circuit Mcc2. (b) The distribution of dead vias after wire pushing of circuit Mcc2.



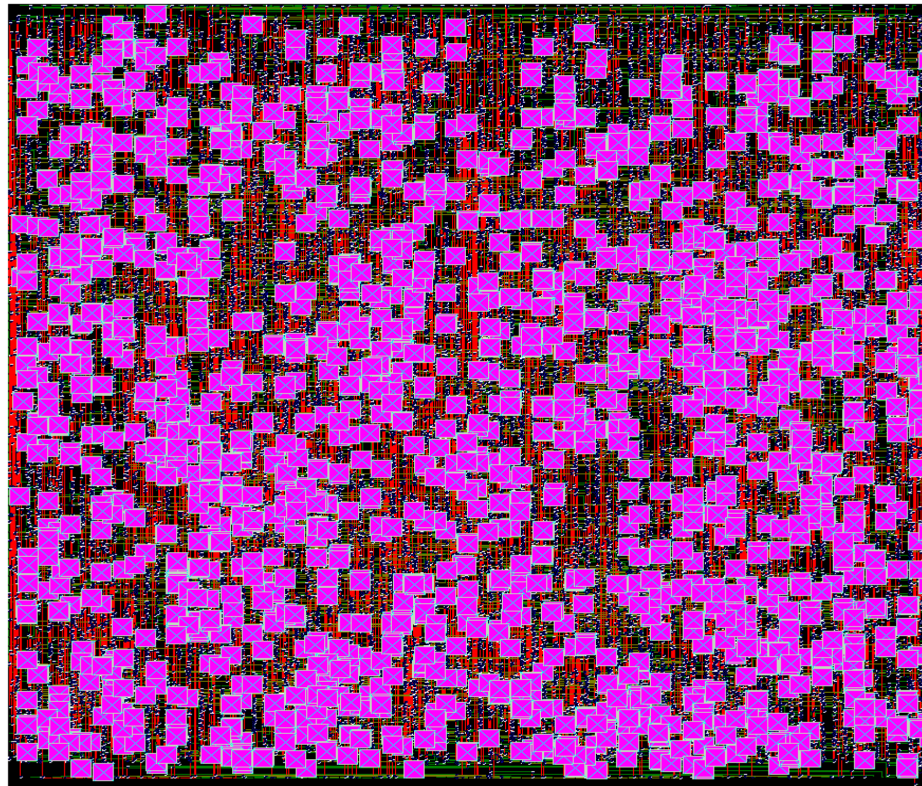


Fig. 4.4: (a) The distribution of dead vias of circuit S38584. (b) The distribution of dead vias after wire pushing of circuit S38584.

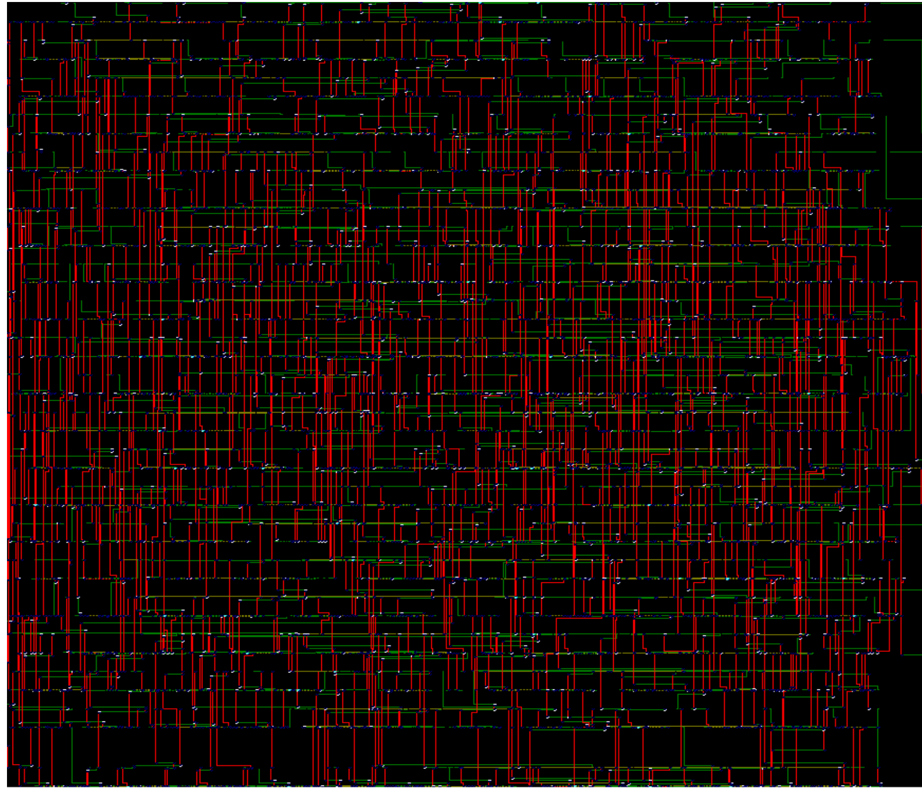


Fig. 4.5: (a) The original routing result of circuit Struct. (b) The distribution of pushing wire segments of circuit Struct.

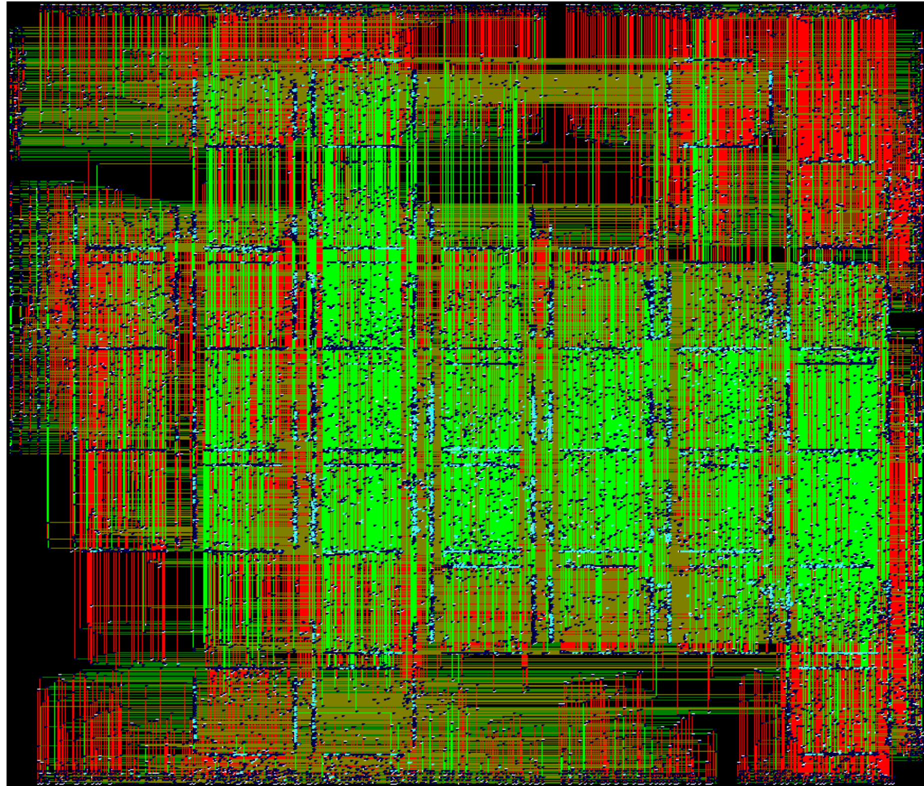


Fig. 4.6: (a) The original routing result of circuit Mcc2. (b) The distribution of pushing wire segments of circuit Mcc2.

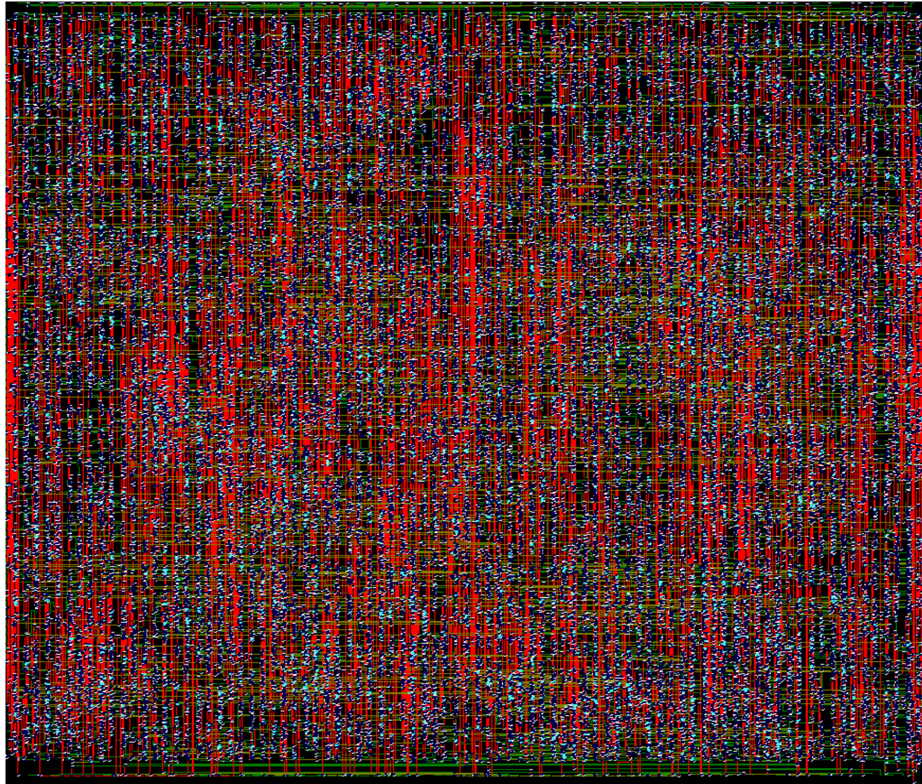


Fig. 4.7: (a) The original routing result of circuit S38584. (b) The distribution of pushing wire segments of circuit S38584.

Chapter 5

Conclusion

In this thesis, the redundant via insertion problem is formulated as a developed MBC graph matching problem, and a heuristic minimum weighted matching (HMWM) algorithm is proposed to solve this matching problem. Then, we utilized the HMWM algorithm and the wire pushing technique to develop an efficient redundant via insertion method for alive and dead vias. The experimental results have shown that our proposed methods can obtain a high redundant via insertion rate for alive vias and protect dead vias efficiently.

Bibliography

[1] Brennan and T. Charles, “Method for Adding Redundant Vias on VLSI Chips”, International Business Machines Corporation, US Patent issued on March 23, 2004.

[2] V. Chowdhury, I. Rahim, A. Yu and G. Venkitachalam, “Translating Yield Learning into Manufacturable Designs”, Altera Corporation, in *Proceedings of The International Symposium for Testing and Failure Analysis*, 2006, pp. 402.

[3] J. Wilson and W. Ng, “Via Doubling to Improve Yield,” Mentor Graphics, Design to Silicon White Paper, 2005.

[4] G. A. Allan, A. J. Walton and R. J. Holwill, “A Yield Improvement Technique for IC Layout using Local Design Rules,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* vol. 11, 1992, pp. 1355–62.

[5] G. A. Allan, A. J. Walton, and R. J. Holwill, “Automated Redundant Via Placement for Increased Yield and Reliability,” in *Proceedings of The International Society for Optical Enginerring*, 1997, vol. 3216, pp. 114–25.

[6] N. Harrison, “A Simple Via Duplication Tool for Yield Enhancement,” in *Proceedings of The IEEE International Symposium on Defect and Fault Tolerance in VLSI*, 2001, pp. 39–47.

[7] Joe G. Xi, “Improving Yield in RTL-to-GDSII Flows,” CMP Media LLC. EE Times: Design News, 2005.

[8] G. A. Allan, “Targeted Layout Modifications for Semiconductor Yield/Reliability Enhancement”, *IEEE Transactions on Semiconductor Manufacturing*, 2004, vol. 17, pp. 573–81.

[9] K. Y. Lee and T. C. Wang, "Post-Routing Redundant Via Insertion for Yield/Reliability Improvement," in *Proceedings of Asia and South Pacific Design Automation Conference*, 2006, pp. 303–8.

[10] K. Y. Lee, T. C. Wang and K.-Y. Chao, "Post-Routing Redundant Via Insertion and Line End Extension with Via Density Consideration," in *Proceedings of International Conference on Computer-Aided Design*, 2006, pp. 633–40.

[11] F. Luo, Y. Jia and W.-M. Dai, "Yield-Preferred Via Insertion Based on Novel Geotopological Technology," in *Proceedings of Asia and South Pacific Design Automation Conference*, 2006, pp. 730–5, .

[12] G. Xu, Li-Da Huang, D. Z. Pan and M. D. F. Wong, "Redundant-Via Enhanced Maze Routing for Yield Improvement," in *Proceedings of Asia and South Pacific Design Automation Conference*, 2005, pp. 1148–51.

[13] H. Yao, Y. Cai, X. Hon and Q. Zhou, "Improved Multilevel Routing with Redundant Via Placement for Yield Improvement," in *Proceedings of Great Lakes Symposium on VLSI*, 2005, pp. 143–6.

[14] H.Y. Chen, M.F. Chiang, Y. W. Chang, L. Chen and B. Han, "Novel Full-Chip Gridless Routing Considering Double-Via Insertion," in *Proceedings of Design Automation Conference*, 2006, pp. 755–60.

[15] 2007 IC/CAD Contest Problem B2: "Double-Via Insertion with Layout Pushing Capability", in *http://www.ee.ncu.edu.tw/∼cad_contest/Problems/95/PB2/2007_B2.pdf*, SpringSoft Inc. 2006.

[16] W.F Brinkman and M.R Pinto, "Future of Solid State Electronics," Bell Labs Technical Journal Autumn, 1997

[17] K. McCullen, "Redundant Via Insertion in Restricted Topology Layouts", in *Proceedings of International Symposium on Quality of Electronic Design*, pp.821–8, 2007.

[18] K.Y. Lee, T.C. Wang and K.Y. Chao, “Optimal Redundant Via Insertion using Mixed Integer Linear Programming,” in *Proceedings of VLSI Design/CAD Symposium*, 2007, pp. 310–3.

[19] K.Y. Lee, C.K. Koh, T.C. Wang and K.Y. Chao, “Optimal Post-Routing Redundant Via Insertion,” in *Proceedings of International Symposium on Physical Design*, 2008, pp. 111–7.

[20] Douglas B. West, *Introduction to Graph Theory*, 2nd ed. Prentice-Hall, 2001.

[21] K. Mehlhorn and S. Näher, *LEDA: A Platform for Combinatorial and Geometric Computing*, Cambridge University Press, 1999.

[22] LEDA. *http://www.algorithmic-solutions.com/leda/*

[23] T.H. Cormen, C.E. Leiseron and R.L. Rivest, *Introduction to Algorithms*, 2nd ed. The Massachuserts Institute of Technology Press, 2001.