Post-Routing Redundant Via Insertion for Yield/Reliability Improvement*

Kuang-Yao Lee and Ting-Chi Wang
Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
d924347@oz.nthu.edu.tw, tcwang@cs.nthu.edu.tw

Abstract - Reducing the yield loss due to via failure is one of the important problems in design for manufacturability. A well known and highly recommended method to improve via yield/reliability is to add redundant vias. In this paper we study the problem of post-routing redundant via insertion and formulate it as a maximum independent set (MIS) problem. We present an efficient graph construction algorithm to model the problem, and an effective MIS heuristic to solve the problem. The experimental results show that our MIS heuristic inserts more redundant vias and distributes them more uniformly among via layers than a commercial tool and an existing method. The number of inserted redundant vias can be increased by up to 21.24%. Besides, since redundant vias can be classified into on-track and off-track ones, and on-track ones have better electrical properties, we also present two methods (one is modified from the MIS heuristic, and the other is applied as a post processor) to increase the amount of on-track redundant vias. The experimental results indicate that both methods perform very well.

I. Introduction

With the advent of the very deep submicron (VDSM) technologies, the process variations become more and more serious, and thus achieving high yield rates on semiconductor chips will be more difficult. In order to reduce the burden of manufacturers to maintain the manufacturability and high yield rates, a new design methodology, design for manufacturability (DFM), is suggested. This design methodology proposes that in order to improve the manufacturability and yield of a design, the manufacturability issues could be considered during the physical design stage [1].

In an IC layout, a via provides a connection between two net segments from adjacent metal layers. Due to the growing of the design scale and/or the jumper-based solution to avoid the antenna effect [11], the number of vias could become very large. However, due to various reasons such as cut misalignment in a manufacturing process, electromigration and thermal stress, a via may fail partially or completely. For a partially failed via, the contact resistance and the parasitic capacitance will increase and may induce timing problems. On the other hand, a complete via failure will leave an open net on the circuit. These may heavily impact the functionality and yield of a design. Therefore, reducing the yield loss due to via failure is one of the most important problems in DFM.

A well known and highly recommended method to improve via yield is to add a redundant via adjacent to a single via [2,3], enabling a single via failure to be tolerated. Therefore, redundant vias will improve the reliability of a design.

Although major EDA vendors have already added the redundant via insertion feature to their routers, their results still have space to improve. (The details will be discussed in section VI.) The tools EYE/PEYE [4] reported in the literature are designed specially to insert redundant vias in the post layout stage but the details of how they do redundant via addition are not given. Besides, according to [4] and the results of the commercial tool used in our experiments, redundant vias are not evenly added on via layers.

[5] is the first work to consider redundant via insertion during the routing stage, but it will overcount the number of alive vias when all alive vias are critical, and cannot estimate the number of free neighbors of alive vias accurately in the general case. (In [5], the number of free neighbors of a via is the number of redundant vias that can be inserted adjacent to the via without inducing any design rule violation; a via with at least one free neighbor is called an alive via.) [6] simultaneously considers redundant via insertion and via minimization during routing. However, in order to reduce the number of vias, the routed wire segments could become longer and violate the antenna rules, and thus need to introduce more vias to fix antenna problems in the post-routing stage. Besides, post-routing ECO operations might also change the routing result and introduce extra vias into the design. Therefore, no matter whether the router considers the redundant via insertion issue or not, it is usually necessary to consider redundant via insertion after detailed routing to improve yield and reliability.

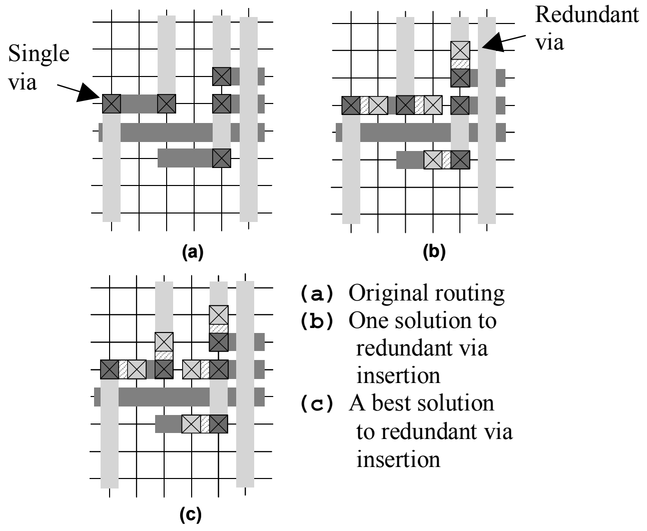
Single via
Redundant via
(a)
(b)
(c)
(a) Original routing
(b) One solution to redundant via insertion
(c) A best solution to redundant via insertion

Fig. 1. Illustration of redundant via insertion.

Given a detailed routing solution, because the positions of inserted redundant vias will affect the number of redundant vias that can be inserted into the design, how to decide the position of each inserted redundant via after detail routing is an important problem. As shown in Fig. 1, we can see that there are only four redundant vias inserted in (b), but as illustrated in (c), all of five single vias can be inserted with redundant vias.

Therefore, in this paper we study the post-routing redundant via insertion problem, and our contributions are threefold. First, we reduce the problem into the maximum independent set problem. All the vias of a circuit are considered simultaneously, and we believe that doing this can get better results than considering redundant via insertion layer by layer. Second, we present an efficient algorithm to construct the conflict graph (to model the problem) from a given detailed routing solution, and an effective heuristic to find a maximal independent set of the graph. The experimental results show that our MIS heuristic not only can insert more redundant vias but also can make the inserted redundant vias more evenly distributed among via layers, as compared to a commercial tool and a method based on [6]. Third, since redundant vias can be classified into on-track and off-track ones, and on-track ones have better electrical properties, we also propose two methods (one is modified from the MIS heuristic, and the other is applied as a post processor) to increase the amount of on-track redundant vias. The experimental results indicate that both methods perform very well.

The rest of this paper is organized as follows. In section II we show that the redundant via insertion problem can be transformed into the maximum independent set problem. In section III, an

* The work was partially supported by the National Science Council of Taiwan under Grant No. NSC-94-2220-E-007-015, and by the Ministry of Economic Affairs of Taiwan under Grant No. MOEA-94-EC-17-A-01-S1-031.

algorithm for constructing the conflict graph from a given detailed routing solution is presented, and then we describe a heuristic method for solving the maximum independent set problem on the conflict graph in section IV. In section V, the methods for increasing the ratio of on-track redundant vias are presented. Section VI gives experimental results, and we conclude the paper in section VII.

II. Problem Formulation

A. Technology

We assume that the manufacturing technology used in this paper consists of $2m+1$ layers denoted by ME_1, VIA_1, ME_2, VIA_2, …, ME_m, VIA_m, ME_{m+1}, where for all i and j, $1 \le i \le m+1$ and $1 \le j \le m$, ME_i and VIA_j represent the ith metal layer and the jth via layer, respectively. A via on VIA_i involves the layers ME_i, VIA_i, and ME_{i+1}. We also assume that a set of design rules is given, and SP is the spacing between two metals or cuts[1].

B. Double vias

The redundant via insertion process is to add a redundant via adjacent to a single via without violating any design rule. For simplicity we name the single via and the inserted redundant via adjacent to it as a *double via.* According to the position of a redundant via, we can categorize a double via into four types, as shown in Fig. 2; a single via is illustrated in (a) and its position is defined at its center; (b), (c), (d) and (e) are the illustrations of the four different double vias, and their types are named *DVU*, *DVD*, *DVL*, and *DVR*, respectively. Given a single via *i*, its double via of type *j* ($j \in \{DVU, DVD, DVL, DVR\}$) is denoted by $dv(i,j)$. For each single via, it has four choices to insert a redundant via if they do not violate any design rule.

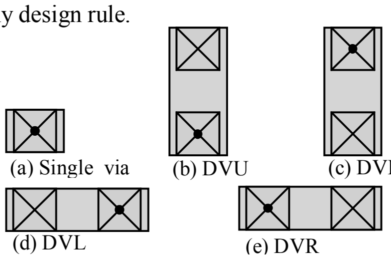

Fig. 2. Double via types.

Definition 1. *(Feasible double via)*

A double via of a single via is said to be feasible if replacing the single via with the double via will not violate any design rule, assuming none of the other single vias has a redundant via inserted in the design; otherwise the double via is defined as an infeasible one.

C. Post-routing redundant via insertion

With the definition of a double via, the post-routing redundant via insertion problem is defined as follows.

Problem 1. *Given a detailed routing solution, without re-routing any signal net, the problem asks to replace single vias on signal nets with double vias as many as possible subject to the following conditions: First, each single via either remains unchanged or is replaced by a double via. Second, after double via replacement, no design rule is violated.*

In the next two subsections, we will discuss two possible formulations, maximum bipartite matching and maximum independent set, to model Problem 1, and explain why the formulation of maximum bipartite matching might not work.

C1. Maximum bipartite matching formulation

[6] reports that Problem 1 can be easily formulated as a maximum bipartite matching problem but without giving any further details. However, we find that either the formulation cannot capture optimal solutions, or some maximum bipartite matchings do not satisfy design rules. We use Fig. 3 to explain it.

Fig. 3(a) gives two different nets, and Fig. 3(b) is their 3D illustrations. Fig. 3(c), (d) and (e) show feasible double vias D_1, D_3 and D_2 for single vias V_1, V_3 and V_2, respectively. Assume that the double vias D_1 and D_2 will introduce some design rule violations if they both exist in the design, and so do the double vias D_3 and D_2. However, because D_1 and D_3 belong to the same net, they can both exist in the design, as shown in Fig. 3(f).

We now describe how to formulate this example as a maximum bipartite matching problem. We construct the bipartite graph $G = (V, E)$ as follows, where $V=X \cup Y$ and there is no edge between any two vertices in X (or between any two vertices in Y). Each single via corresponds to a vertex in X. Each feasible double via corresponds to at least one edge in E. For two feasible double vias originating from different single vias, if their existence in the design will violate design rules, their corresponding edges in E will be incident to the same vertex in Y. Fig. 3(g), (h) and (i) are three possible bipartite graphs obtained from this formulation. In graph Fig. 3(g) or (h), the set of bold edges is a maximum bipartite matching solution. However, neither of them is a legal solution to this example. On the other hand, the bipartite graph shown in Fig. 3(i) does not include the optimal solution to this example. We are not aware of any other way to construct the bipartite graph, but at least the three ones shown in Fig. 3(g), (h) and (i) cannot model Problem 1 correctly.

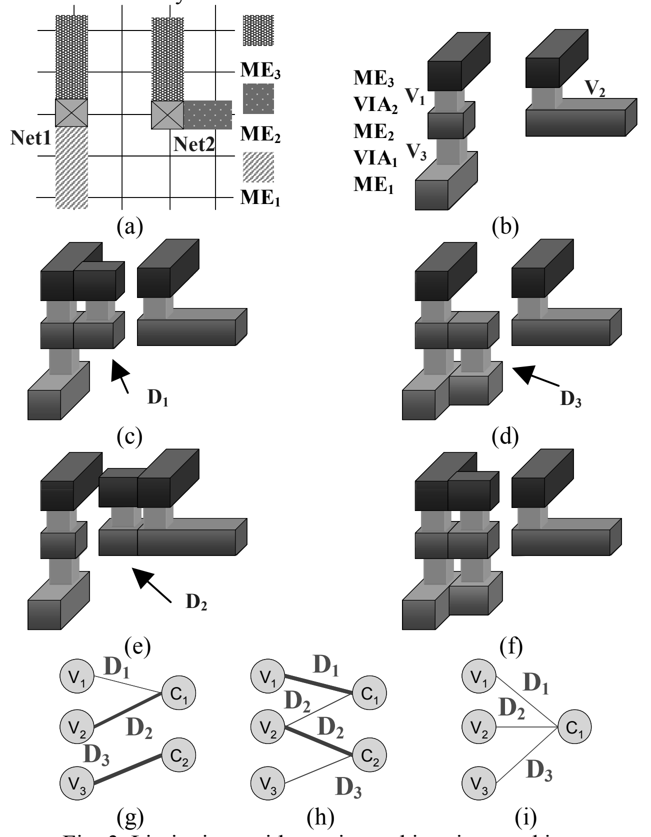

Fig. 3. Limitations with maximum bipartite matching.

C2. Maximum independent set formulation

Before introducing the maximum independent set formulation, we need to define what a conflict graph is first.

Definition 2. *(Conflict graph)*

A conflict graph G(V,E) is an undirected graph constructed from a detailed routing solution. For each single via i on a signal net, if its double via of type j (i.e., dv(i,j)) is feasible, there exists a vertex $v_{i,j}$ in V. An edge $(v_{i,j}, v_{i',j'}) \in E$ if and only if $i=i'$, or dv(i,j) and

[1] Depending on the technology, the spacing between metals could be different from the spacing between cuts. Also these space rules could vary on different layers. Nevertheless, our redundant via insertion methods presented in this paper can be easily modified to handle all these cases.

dv(i',j') will cause design rule violations when both exist in the design.

Lemma 1. *Problem 1 can be reduced into the maximum independent set problem.*

Proof. Consider the conflict graph $G(V,E)$ constructed from a routed design. A maximum independent set MV of G is a maximum vertex set such that, $\forall\, v_{i,j}, v_{i',j'} \in MV$, $(v_{i,j}, v_{i',j'}) \notin E$. A vertex of G represents a feasible double via, and if two vertices are the endpoints of an edge, the corresponding double vias will violate design rules or they come from the same single via. Hence a maximum independent set of G is a set having the maximum number of double vias that can be inserted into the design. ■

With Lemma 1, Problem 1 can be reduced to the following problem.

Problem 2. *Given a detailed routing solution, the problem asks to first construct a conflict graph from the design, then find a maximum independent set of the conflict graph, and finally for each vertex $v_{i,j}$ in the maximum independent set, replace the single via i with the double via dv(i, j).*

In the following two sections, we will describe how to efficiently construct a conflict graph and find a maximal independent set of the conflict graph.

III. Conflict Graph Construction

The construction of a conflict graph can be briefly divided into the vertex construction step and the edge construction step.

For the vertex construction step, we have to identify the feasible double vias of each single via. First, under the consideration of time complexity, we construct an R-tree [7,8,9] for each metal layer instead of constructing a single R-tree for all metal layers. An R-tree and its variants are data structures that are similar to a B-tree, but are used for indexing multi-dimensional information. In this paper, we use an R-tree for indexing 2-dimensional information. Typical queries on an R-tree specify a window of interest and retrieve all data intersecting or contained in the specified query window.

For a metal layer, the corresponding R-tree consists of the bounding box[2] of each object such as a wire segment, pin, or obstacle on the layer; besides, the bounding box of the vias on adjacent via layers are also included in the R-tree.

Definition 3. *(DVE)*

Suppose the bounding box of a single via i is $R_i=[x_{11},x_{12}]\times[y_{11},y_{12}]$, where (x_{11},y_{11}) and (x_{12},y_{12}) are the coordinates of the lower left corner and the upper right corner of the bounding box, respectively (see Fig. 4(a)); suppose the bounding box of a double via dv(i,j) is $R_{dv(i,j)} =[x_{21},x_{22}]\times[y_{21},y_{22}]$ (see Fig. 4(b)). The reduced bounding box of dv(i,j), denoted by DVE(i,j), is defined as $R_{dv(i,j)}-R_i=[x_{e1},x_{e2}]\times[y_{e1},y_{e2}]$ (see Fig. 4(c) for the illustration of DVE(i,DVU)).

Definition 4. *(DRW)*

Given a double via dv(i,j), suppose the bounding box of the redundant via contained in dv(i, j) is $R_{rv}=[x_{r1},x_{r2}]\times[y_{r1},y_{r2}]$. Then, the reduced design rule window of dv(i,j) is defined to be DRW(i,j) $=[x_{r1}-SP,\ x_{r2}+SP]\times[y_{r1}-SP,\ y_{r2}+SP]$. (See Fig. 4(d) for the illustration of DRW(i,DVU) which is the region with oblique lines.)

Definition 5. *(DRWSET and DVESET)*

The DRW set and DVE set of a single via i, denoted by DRWSET(i) and DVESET(i), are defined to be {DRW(i,j) | dv(i,j) is a feasible double via} and {DVE(i,j) | dv(i,j) is a feasible double via}, respectively.

[2] The bounding box of an object in the design is the contour of its 2-dimensional structure.

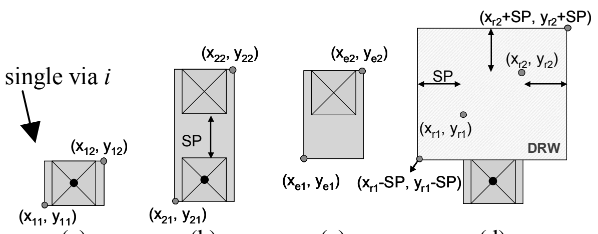

Fig. 4. Illustration of the *DVE* and the *DRW* for DVU.

For the vertex construction step, since a vertex in the conflict graph corresponds to a feasible double via, we need to check each double via and decide if it is feasible. In the following, we describe the details of the vertex construction step.

A. Vertex set construction

For each double via $dv(i,j)$ originating from a single via i on layer VIA_k, we construct $DRW(i,j)$ and use it as a query window to perform the range query on R-trees of ME_k and ME_{k+1}.

If there are any objects intersecting with $DRW(i,j)$, we cannot replace single via i with $dv(i,j)$, because it will induce design rule violations. Hence, there will never be a vertex on the conflict graph for $dv(i,j)$. On the other hand, if there is no object intersecting with $DRW(i,j)$, we add a vertex $v_{i,j}$ to the conflict graph.

After constructing the vertex set of a conflict graph, we should start the edge construction step. However, if we construct the edges of the conflict graph after completing the vertex construction step, the time complexity will be $O(n^2)$, where n is the number of vertices in the conflict graph. In fact, we can construct the edges more efficiently by constructing the vertex and edge sets simultaneously, as detailed in following subsection.

B. Graph construction algorithm

If there is an edge connecting two vertices in a conflict graph, the two double vias corresponding to the ends of the edge will belong to the same single via, or induce some design rule violations if they both exist in the design. Furthermore, a double via may introduce design rule violations to another one only if their corresponding single vias locate in nearby grids. Therefore, we first sort all single vias by their x-coordinates in the non-decreasing order. We then construct an R-tree for each metal layer, and finally according to the sorted order of vias (denoted $1, 2, \ldots, n$), we perform the following four steps (i.e., Step 1 through Step 4) for each single via to get the conflict graph.

Before stating the details of the graph construction algorithm (called *GCA*), we introduce another R-tree named *VNC* first. *VNC* consists of the *DVE*s of feasible double vias, and initially it is empty. Once a single via *i* has been processed, each element of *DVESET*(*i*) will be inserted into *VNC*. For each element of *VNC*, if it will never intersect with any element of *DRWSET*(*j*), for those sigle via *j*'s that have not been processed, it will be deleted from *VNC*. With *VNC*, we can construct edges efficiently.

Step 1. Suppose i is the single via being under consideration and x_{ll} is the x-coordinate of the lower left corner of the bounding box of i. If i is located in (x_i,y_i) and none of the x-coordinates of single vias $1, 2, \ldots, i-1$ is equal to x_i, we retrieve the elements of *VNC* contained in the range $[-\infty, x_{ll}-SP]\times[-\infty,+\infty]$ and delete them from *VNC*, since these elements will never overlap with any element of $DRWSET(j)$ with $j \geq i$.

Step 2. We start the vertex construction step for i. For each $dv(i,j)$, we use $DRW(i,j)$ as the query window to do the range query on the R-trees of adjacent metal layers (Details are as described in the previous subsection.). Suppose the set of added vertices for i is called $FV(i)$.

Step 3. First, we add an edge for each vertex pair of $FV(i)$ to the conflict graph. Then, we use each element of $DRWSET(i)$ as the query window to do the range query on *VNC*. For each vertex $v_{i,j} \in FV(i)$, we can get a vertex set V', where for each $v_{i',j'} \in V'$, the corresponding element in *VNC*, i.e., $DVE(i',j')$, intersects with $DRW(i,j)$. However, we cannot directly add an edge $(v_{i,j},v_{i',j'})$ to the conflict graph, because $v_{i,j}$, $v_{i',j'}$ may belong to the same net. Therefore, we need to check each pair $(v_{i,j},v_{i',j'})$ to see if they really introduce any design rule violation.

Step 4. We insert each element of *DVESET*(i) to *VNC*.

Note that in Step 3, we need to check each pair $(v_{i,j},v_{i',j'})$ to see if they really introduce any design rule violation, because there are cases where even if *DVE*(i',j') intersects with *DRW*(i,j), inserting both double vias *dv*(i,j) and *dv*(i',j') into the design still will not violate any design rule. A possible case is depicted in Fig. 5, where single vias V_1 and V_2 belong to the same net. In Fig. 5(b), *DVE*(V_1, *DVR*) (*DVE*(V_2,*DVL*), respectively) intersects with *DRW*(V_2,*DVL*) (*DRW*(V_1,*DVR*), respectively). However, they will not violate any design rule because they belong to the same net.

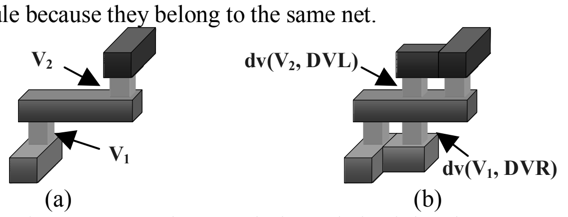

Fig. 5. A case where no design rule is violated.

Fig. 6 illustrates how the graph construction algorithm *GCA* works. In Fig. 6(a), there are four single vias in the design, and they are numbered to form the sorted sequence. After processing via 2, the elements of *VNC* and the conflict graph are shown in Fig. 6(b). When processing via 3, suppose *DVE*($1,DVU$) and *DVE*($1,DVR$) are contained in $[-\infty, x_{3,ll}-SP]\times[-\infty,+\infty]$, where $x_{3,ll}$ is the x-coordinate of the lower left corner of the bounding box of via 3. Therefore, after Step 1, *DVE*($1,DVU$) and *DVE*($1,DVR$) are deleted from *VNC* and the remaining elements in *VNC* are shown in Fig. 6(c). Then, after Step 2, the conflict graph gets updated as shown in Fig. 6(c). Finally, after Steps 3 and 4, *VNC* and the conflict graph become those shown in Fig. 6(d). Via 4 will be processed similarly.

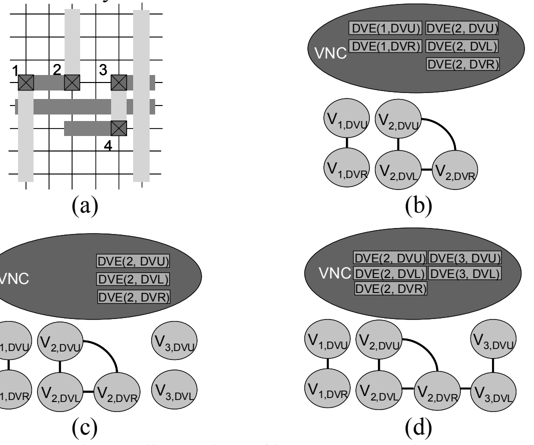

Fig. 6. Illustration of how *GCA* works.

IV. Heuristic for Solving the MIS Problem

Now we present a heuristic that solves the maximum independent set (MIS) problem on the conflict graph.

It is well known that the MIS problem is an NP-hard problem, so it is unlikely that we can get an optimal solution in polynomial time. Besides, the time complexities of MIS solvers are usually growing very fast as the numbers of vertices and edges in the graph increase. Therefore, our heuristic (called *H2K*) will solve the MIS problem in an iterative manner. In each iteration, a subgraph of size k (which specifies the maximum number of vertices in the subgraph and is a user-specified constant) is extracted from the conflict graph, a maximal independent set solution to the subgraph is sought and added to the final solution, and the conflict graph is updated. *H2K* will terminate when the conflict graph has no remaining vertices.

Before describing the details of *H2K*, we define the "feasible number" for each vertex. The *feasible number* of each vertex $v_{i,j}$ in the conflict graph is equal to the number of vertices $v_{i',j'}$'s (excluding $v_{i,j}$ itself) in the conflict graph such that $i=i'$ (i.e., the number of the other feasible double vias originating from the same single via). Initially, the feasible number of each $v_{i,j}$ is equal to $|DVESET(i)|$-1, where $|DVESET(i)|$ is the cardinality of *DVESET*(i). The feasible number and degree of each vertex will decrease during the execution of *H2K*. The detailed steps of *H2K* are given as follows.

Step 1. For the conflict graph $G(V,E)$, we construct a priority queue Q of V by using the feasible number and degree of a vertex as the first and second keys. We give a vertex a higher priority if it has smaller feasible number and degree. In addition, we define a vertex set V_{sol} to be the maximal independent set solution to G. Initially V_{sol} is an empty set.

Step 2. We extract the set $V_{sub}=\{v_1,v_2,\ldots,v_k\}$ of the first k vertices from Q, and construct the graph $G'=(V_{sub}, E')$, where $\forall\, v_i, v_j \in V_{sub}$, $(v_i,v_j)\in E'$ if $(v_i,v_j)\in E$.

Step 3. Solve the MIS problem on G' and get the solution denoted V_{tsol}.

Step 4. We set $V_{sol} = V_{sol}\cup V_{tsol}$ and then delete the vertices of V_{tsol} and their adjacent vertices from G and Q. Moreover, each edge incident to any deleted vertex is also removed from G. Finally, we update the feasible number and degree of each remaining vertex which is originally adjacent to some deleted vertex. In addition, Q is also updated.

Step 5. If V is empty, the vertex set V_{sol} is our final solution; otherwise we go back to Step 2.

The rationale behind subgraph extraction (i.e., Step 2) is that if a vertex with smaller feasible number and degree appears in the maximal independent set solution to G', less vertices will be deleted from the conflict graph in Step 4. Therefore, we prefer solving the MIS problem on a subgraph containing vertices with smaller feasible numbers and degrees.

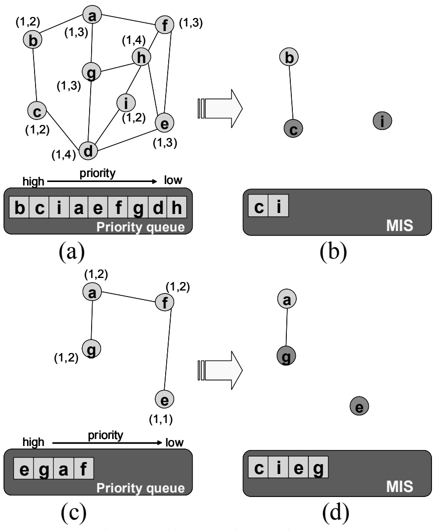

Fig. 7. Illustration of *H2K*.

Fig. 7 illustrates how our *H2K* works, where each vertex is attached with a pair of numbers; the first number is the feasible number, and the second number is the degree. To simplify the example, we assume the feasible number of each vertex is equal to one. In the beginning, the conflict graph G and the priority queue Q are shown in Fig. 7(a). In Step 2, suppose k is set to 3, and the extracted subgraph G' has the vertex set $\{b, c, i\}$ as shown in Fig. 7 (b). Suppose the maximal independent set solution to G' found in Step 3 is $\{c, i\}$. Then in Step 4, G and Q are updated by deleting vertices c, i, and their adjacent vertices; each edge incident to any deleted vertex is also removed from G. The resultant G and Q are shown in Fig. 7 (c). At the second iteration, G' will be the one shown in Fig. 7 (d) and the maximal independent set solution to G' is assumed to be $\{g, e\}$. After Step 4 is done, G is empty, and hence the final solution found by *H2K* will be $\{c, e, i, g\}$.

V. On- and Off-track Redundant Vias

As shown in Fig. 8, a redundant via *rv* of a single via *v* is called an *on-track* redundant via if *rv* is inserted on a wire segment connecting to *v*; otherwise, *rv* is called an *off-track* redundant via. Since an on-track redundant via takes less routing resource and has better electrical properties than an off-track redundant via, on-track redundant vias are more preferable. Therefore, if two solutions contain the same number of redundant vias, we prefer the one with more on-track redundant vias.

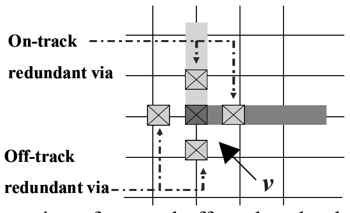

Fig. 8. Illustration of on- and off-track redundant vias.

A double via is said to be *on-track* if its associated redundant via is an on-track redundant via; otherwise it is an *off-track* double via. We now modify Problem 1 to consider the preference of on-track redundant vias as well.

Problem 3. *Given a detailed routing solution, without re-routing any signal net, the problem asks to replace single vias on signal nets with double vias as many as possible, and the ratio of on-track double vias should be also as high as possible. In addition, two conditions should be satisfied. First, each single via either remains unchanged or is replaced by a double via. Second, after double via replacement, no design rule is violated.*

We present two methods to solve Problem 3. The first one is to modify *H2K* by adding the third key to each vertex in the priority queue. If a vertex corresponds to an on-track double via, it will have a higher priority on this key. With this modification, for vertices having the same feasible number and degree, on-track ones will be extracted first, and hence have higher chances to be included in the maximal independent set solution than off-track ones. We call this method as *H3K*.

In addition, we also present a post processing heuristic (called *PPH*). Given a redundant via insertion solution, *PPH* will increase the amount of on-track double vias as many as possible while at the same time without decreasing the total number of double vias. *PPH* works as follows. It takes a conflict graph $G(V,E)$ and a redundant via insertion solution $RVIS_{org}$ as the input, and will generate another vertex set $RVIS_{mod}$ as the output. Initially $RVIS_{mod}$ is an empty set. In addition, a Boolean flag *IS_DEL* is used in *PPH*. Without loss of generality, $RVIS_{org}$ is assumed to be a set of vertices, and we will interchangeably use vertices and double vias. Each vertex *v* of $RVIS_{org}$ will be processed by the following four steps in a random order.

Step 1. Set *IS_DEL* to *FALSE*.

Step 2. If *v* is an on-track double via, go to Step 4. Otherwise, go to Step 3.

Step 3. Check each adjacent vertex *v'* of *v* in *G*. If *v'* is an on-track double via and each adjacent vertex of *v'* (excluding *v*) is not in $RVIS_{org} \cup RVIS_{mod}$, add *v'* to $RVIS_{mod}$ and set *IS_DEL* to *TRUE*.

Step 4. If *IS_DEL* is *FALSE*, *v* will be moved from $RVIS_{org}$ to $RVIS_{mod}$. Otherwise, *v* will be deleted from $RVIS_{org}$.

VI. Experimental Results

The technology used in our experiment has 5 metal layers. For simplicity we directly used the R*-tree package [9] for indexing 2-dimensional information of each metal layer. Moreover, we used the qualex-ms [10] as our MIS solver; we tried many different sizes when extracting a subgraph, and found that if we limited the subgraph to consist of 1500 vertices at most, it could get the best performance in terms of the number of inserted redundant vias.

Table 1: The experimental results on test cases

	Via1	Via2	Via3	Via4	Total	CPU(s)
C1 Statistics						
Original	11979	11111	1462	42	24594	
Upper	5218	10819	1443	42	17522	
CT	2125	10797	1438	42	14402	19
RatC(%)	17.74	97.17	98.36	100	58.56	
FNF	5165	10788	1438	42	17433	34
RatF(%)	43.12	97.09	98.36	100	70.88	
ImpF(%)	143.06	-0.08	00.00	00.00	21.05	
H2K	5175	10803	1441	42	17461	32
Rat2K(%)	43.20	97.23	98.56	100	71.00	
Imp2K(%)	143.53	00.06	00.21	00.00	21.24	
C2 Statistics						
	Via1	Via2	Via3	Via4	Total	CPU(s)
Original	17208	18086	4745	1118	41157	
Upper	6078	17066	4359	1088	28591	
CT	3476	17005	4351	1086	25918	28
RatC(%)	20.20	94.02	91.70	97.14	62.97	
FNF	6059	16982	4325	1085	28451	45
RatF(%)	35.21	93.90	91.15	97.05	69.13	
ImpF(%)	74.31	-0.14	-0.60	-0.90	09.77	
H2K	6069	17011	4341	1086	28507	43
Rat2K(%)	35.27	94.06	91.49	97.14	69.26	
Imp2K(%)	74.60	00.04	-0.23	00.00	09.99	
C3 Statistics						
	Via1	Via2	Via3	Via4	Total	CPU(s)
Original	55878	55252	13066	2863	127059	
Upper	23755	52780	12407	2785	91727	
CT	13179	52506	12365	2777	80827	101
RatC(%)	23.59	95.03	94.63	97.00	63.61	
FNF	23634	52539	12358	2784	91315	190
RatF(%)	42.30	95.09	94.58	97.24	71.84	
ImpF(%)	79.33	00.06	-0.06	00.25	12.98	
H2K	23687	52615	12375	2784	91461	192
Rat2K(%)	42.39	95.23	94.71	97.24	71.98	
Imp2K(%)	79.73	00.21	00.08	00.25	13.16	
C4 Statistics						
	Via1	Via2	Via3	Via4	Total	CPU(s)
Original	57216	64879	20864	8953	151912	
Upper	14917	61300	17950	8180	102347	
CT	4677	60978	17777	8142	91574	120
RatC(%)	08.17	93.99	85.20	90.94	60.28	
FNF	14750	60848	17711	8148	101457	201
RatF(%)	25.78	93.79	84.89	91.01	66.79	
ImpF(%)	215.37	-0.21	-0.37	00.07	10.79	
H2K	14805	61008	17791	8161	101765	203
Rat2K(%)	25.88	94.03	85.27	91.15	66.99	
Imp2K(%)	216.55	00.05	00.08	00.23	11.13	
C5 Statistics						
	Via1	Via2	Via3	Via4	Total	CPU(s)
Original	148661	158862	40726	9137	357386	
Upper	62312	148592	35729	8668	255301	
CT	33216	147781	35505	8640	225142	311
RatC(%)	22.34	93.02	87.18	94.56	63.00	
FNF	62033	147757	35453	8656	253899	697
RatF(%)	41.73	93.01	87.05	94.74	71.04	
ImpF(%)	86.76	-0.02	-0.15	00.19	12.77	
H2K	62174	148063	35535	8656	254428	710
Rat2K(%)	41.82	93.20	87.25	94.74	71.19	
Imp2K(%)	87.18	00.19	00.08	00.19	13.01	

[6] points out a simple heuristic for redundant via insertion and its idea is that if there is only one feasible redundant via for a single via, it adds the redundant via first. However, [6] does not provide any further details. We also based on the above idea and implemented a heuristic called *FNF* for comparative studies. Its details are as follows. *FNF* takes a conflict graph as the input, and creates a priority queue for vertices such that a vertex with smaller feasible number has a higher priority. *FNF* iteratively extracts the vertex with the smallest feasible number from the priority queue, adds it into the final solution, and updates the priority queue and

the conflict graph. When the conflict graph or priority queue is empty, *FNF* terminates.

We first compared our approach *H2K* with a commercial tool and *FNF* on five real circuits C1-C5. Our experimental flow is as follows. We used the commercial tool to generate the routed circuit, and then inserted redundant vias by its redundant via insertion feature. Each conflict graph used by *H2K* and *FNF* was generated by our *GCA* algorithm that took the routed design as the input. Then, *H2K* and *FNF* generated the circuits with inserted redundant vias. Finally, the results obtained by the commercial tool, *H2K* and *FNF* were verified with the built-in DRC and LVS verifier of the commercial tool.

The results are shown in Table 1. “Original” gives the number of single vias on each via layer before performing redundant via insertion. “Upper” denotes the number of single vias that have at least one feasible double via. “CT”, “FNF” and “H2K” are the numbers of redundant vias inserted by the commercial tool, *FNF* and *H2K*, respectively. “RatC(%)”, “RatF(%)” and “Rat2K(%)” are the ratios of “CT”, “FNF” and “H2K” to “Original”, respectively. “ImpF(%)” and “Imp2K(%)” represent the improvement rates of *FNF* and *H2K* over the commercial tool, respectively. “CPU(s)” gives the CPU time in seconds of different approaches. The commercial tool was executed on a Sun Fire V440 machine with four CPUs and 8GB memory; *H2K*, *GCA* and *FNF* were implemented in C++ language running on a Linux based machine with 2.4G processor and 2GB memory. Because *H2K*, *GCA* and *FNF* used some Linux based packages, they could not be executed on a Sun based platform. It should be noted that the CPU times for the commercial tool only record the redundant via insertion step, and before this step the design has been loaded into memory. The CPU times of *H2K* and *FNF* include the time spent by *GCA*.

From Table 1, we can see that our approach *H2K* can insert 9.99%-21.24% more redundant via than the commercial tool. Besides, the number of redundant vias inserted on each layer by *H2K* is very close to the upper bound in all test cases, but the number of redundant via inserted on Via1 by the commercial tool is much smaller than the upper bound. Hence, the redundant vias inserted by *H2K* are distributed more uniformly among via layers. Moreover, the experimental results show that although *FNF* also inserts more redundant vias than the commercial tool, its improvement rate is still less than our approach *H2K* for each test case. *H2K* can insert up to 529 more redundant vias than *FNF* with comparable CPU time. In every test case, there is at least one via layer on which *FNF* inserts less redundant vias than the commercial tool. Nevertheless, our approach *H2K* can always insert more or the same number of redundant vias among each via layer than *FNF* and the commercial tool.

Table 2 shows the results of our approaches *H3K* and *PPH* when considering on-track redundant vias. “FNF+PPH”, “H2K+PPH” and “H3K+PPH” indicate that *PPH* was applied after *FNF*, *H2K*, and *H3K*, respectively. It should be mentioned that although *H3K* is design to consider on-track redundant vias directly, we would like to see if its result still has room to improve, and therefore we also applied *PPH* after *H3K*.

The columns “MISo” and “ONo” show the numbers of double vias and the on-track double vias from each original solution, respectively. After running *PPH,* the numbers of inserted double vias and on-track double vias are shown in the columns “MISm” and “ONm”, respectively. The column “Imp(%)” denotes the improvement rate on the number of on-track double vias achieved by *PPH*. “CPU(s)” gives the CPU time of *PPH*, but for “H3K”, it represents the total CUP time of *H3K*.

From Table 2, we can see that even if we prefer on-track redundant vias, the total number of inserted redundant vias can still remain the same or even larger while the CPU time spent by *PPH* is no more than 3 seconds. Compared to *H2K*, *H3K* can increase the number of on-track double vias by up to 65.31% while almost having the same number of inserted redundant vias and spending the same or less CPU time. As for *PPH*, it helps to increase the amount of on-track double vias by 19.99%-21.90% and 18.58%-20.54% for *FNF* and *H2K*, respectively. Besides, for some test cases, *PPH* can also slightly increase the total number of redundant vias. Finally, we observe that running *H3K* alone is always good enough to beat both “FNF+PHP” and “H2K+PHP” on the number of on-track redundant vias, although its result can still be improved by *PHP* for more than half of the test cases.

Table 2: The experimental results for *H3K* and *PPH*.

C1 Statistics						
	MISo	ONo	MISm	ONm	Imp(%)	CPU(s)
FNF+PPH	17433	7128	17433	8553	19.99	<1
H2K+PPH	17461	7167	17461	8552	19.32	<1
H3K	17461	11848	-	-	-	32
H3K+PPH	17461	11848	17461	11878	00.25	<1
C2 Statistics						
	MISo	ONo	MISm	ONm	Imp(%)	CPU(s)
FNF+PPH	28451	13132	28451	15986	21.73	1
H2K+PPH	28507	13406	28507	16047	19.70	<1
H3K	28506	20508	-	-	-	43
H3K+PPH	28506	20508	28506	20519	00.05	<1
C3 Statistics						
	MISo	ONo	MISm	ONm	Imp(%)	CPU(s)
FNF+PPH	91315	42084	91318	50551	20.12	1
H2K+PPH	91461	42397	91461	50275	18.58	1
H3K	91461	66205	-	-	-	190
H3K+PPH	91461	66205	91461	66212	00.01	1
C4 Statistics						
	MISo	ONo	MISm	ONm	Imp(%)	CPU(s)
FNF+PPH	101457	47649	101459	58084	21.90	1
H2K+PPH	101765	48073	101765	57946	20.54	<1
H3K	101765	70696	-	-	-	201
H3K+PPH	101765	70696	101765	70696	00.00	1
C5 Statistics						
	MISo	ONo	MISm	ONm	Imp(%)	CPU(s)
FNF+PPH	253899	117432	253903	142331	21.20	3
H2K+PPH	254428	118557	254428	142251	19.99	2
H3K	254428	180512	-	-	-	680
H3K+PPH	254428	180512	254428	180513	00.00	1

VII. Conclusions

In this paper we consider the post-routing redundant via insertion problem which is formulated as the maximum independent set problem. We present an efficient graph construction algorithm to model the problem, and an effective heuristic to solve the maximum independent set problem. Besides, we also describe how to modify the MIS heuristic and give a post-processing method to increase the amount of on-track redundant vias. Promising experimental results are shown to support all our methods.

VIII. References

[1] L. K. Scheffer, “Physical CAD Changes to Incorporate Design for Lithography and Manufacturability”, Proc. of ASPDAC, 2004.

[2] TSMC Reference Flow 5.0.

[3] Y. Zorian, D. Gizopoulos, C. Vandenberg and P. Magarshack, “Guest Editors’ Introduction: Design for Yield and Reliability”, IEEE Trans on Design & Test of Computers, vol. 21, May 2004.

[4] G. A. Allan, “Targeted Layout Modifications for Semiconductor Yield/Reliability Enhancement”, IEEE Trans on Semiconductor Manufacturing, vol. 17, Nov. 2004.

[5] G. Xu, Li-Da Huang, D. Z. Pan and M. D. F. Wong, “Redundant-Via Enhanced Maze Routing for Yield Improvement”, Proc. of ASPDAC, 2005.

[6] H. Yao, Y. Cai, X. Hong and Q. Zhou, “Improved Multilevel Routing with Redundant Via Placement for Yield and Reliability”, Proc. of GLSVLSI, 2005.

[7] A. Guttman, “R-Trees: A Dynamic Index Structure for Spatial Searching”, Proc of SIGMOD, 1984.

[8] M. de Berg, J. Gudmundsson, M. Hammar and M. H. Overmars, “On R-trees with Low Stabbing Number”, Proc. European Symposium on Algorithms, 2000.

[9] N. Beckmann, H.-P. Kriegel, R. Schneider and B. Seeger, “The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles”, Proc. of SIGMOD, 1990.

[10] http://www.busygin.dp.ua/npc.html

[11] P. H. Chen, S. Malkani, C.-M. Peng and J. Lin, “Fixing Antenna Problem by Dynamic Diode Dropping and Jumper Insertion”, Proc. of ISQED, 2000.