Post-Routing Redundant Via Insertion and Line End Extension with Via Density Consideration *

Kuang-Yao Lee
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
d924347@oz.nthu.edu.tw

Ting-Chi Wang
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
tcwang@cs.nthu.edu.tw

Kai-Yuan Chao
Intel Corporation
Hillsboro, OR 97124
kaiyuan.chao@intel.com

Abstract - Redundant via insertion and line end extension employed in the post-routing stage are two well known and highly recommended techniques to reduce yield loss due to via failure. However, if the amount of inserted redundant vias is not well controlled, it could violate via density rules and adversely worsen the yield and reliability of the design. In this paper, we first study the problem of redundant via insertion, and present two methods to accelerate a state-of-the-art approach (which is based on a maximum independent set (MIS) formulation) to solve it. We then consider the problem of simultaneous redundant via insertion and line end extension. We formulate the problem as a maximum weighted independent set (MWIS) problem and modify the accelerated MIS-based approach to solve it. Lastly, we investigate the problem of simultaneous redundant via insertion and line end extension subject to the maximum via density rule, and present a two-stage approach for it. In the first stage, we ignore the maximum via density rule, and enhance the MWIS-based approach to find the set of regions which violate the maximum via density rule after performing simultaneous redundant via insertion and line end extension. In the second stage, excess redundant vias are removed from those violating regions such that after the removal, the maximum via density rule is met while the total amount of redundant vias removed is minimized. This density-aware redundant via removal problem is formulated as a set of zero-one integer linear programming (0-1 ILP) problems each of which can be solved independently without sacrificing the optimality. The superiorities of our approaches are all demonstrated through promising experimental results.

I. Introduction

As the manufacturing technology shrinks, the feature size of a layout object becomes smaller but the scale of an integrated circuit (IC) becomes larger. However, the process variation becomes worse and damages the yield of an IC. In order to maintain manufacturability and high yield rates, a new design methodology, called design for manufacturability (DFM), is suggested [10][14]. To reduce the yield loss due to via failures is one of the most important issues in DFM.

A via in an IC layout provides the connection between two net segments from adjacent metal layers. The number of vias could become very large due to the design scale growing and/or the advent of the jumper-based solution to avoid the antenna effect [12]. Vias may fail partially or completely due to various reasons, such as cut misalignment and/or line-end shortening [14] during manufacturing processes. For a partially failed via, the contact resistance and the parasitic capacitance will increase and cause unexpected delay. On the other hand, a complete via failure will leave an open net in an IC layout and invalidate the functionality of the design.

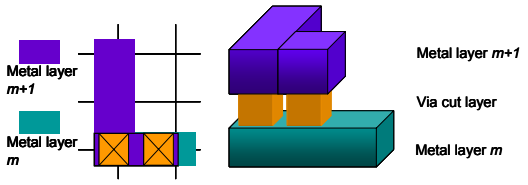


Fig. 1 Illustration for redundant via insertion.

One of the well known and highly recommended method to improve via yield/reliability is to add a redundant via adjacent to a single via [15][17]. Fig. 1 shows the top view and the 3D structure of a single via with a redundant via added to its right side. When a single via fails, its redundant via may still work; besides, the redundant via also provides an alternative signal path. Therefore, after adding a redundant via, the single-via failure can be tolerated and the whole via resistance can be also reduced.

The redundant via insertion problem can be considered in the routing or post-routing stage. The tools EYE/PEYE [5] consider redundant via insertion in the post-routing stage but the details of how they do redundant via insertion are not described. [13] and [4] also consider redundant via insertion in the post-routing stage. In [4], the single vias of a design are considered one by one to perform redundant via addition, and therefore the solution may just be locally optimal. Besides, since the approach will change the routing result of the timing non-critical nets, it may also induce timing violations even if designers keep the timing critical nets unchanged. [13] reduces the post-routing redundant via insertion problem into the maximum independent set (MIS) problem and proposes an effective heuristic to solve the MIS problem. The execution time of the approach, however, is generally longer according to the results reported in [13].

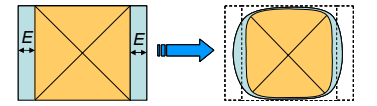
Both [6] and [7] consider redundant via insertion in the routing stage. [6] proposes a Lagrangian relaxation-based solution and [7] extends an existing multi-level routing framework to consider via minimization as well. However, post-routing ECO operations may change routing results and introduce extra vias into designs for the purposes of fixing timing, antenna or other problems. Therefore, no matter whether a router considers the redundant via insertion issue or not, it is usually necessary to perform redundant via insertion after the routing stage to further improve the yield and reliability of vias.

Among the set of via related design rules, the via density rules belong to the category of density control rules that arise for chemical-mechanical polishing (CMP) and other manufacturing steps which have varying effects on device and interconnect features depending on local layout density characteristics [1][16]. For each area of a pre-defined size on a via layer, its via density can be defined as the number of vias within it. If too many redundant vias are inserted into a die area, and exceed the maximum via density constraint, the pattern

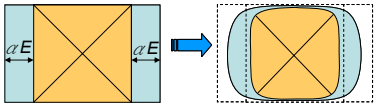
* This work was partially supported by National Science Council under Grant No. NSC-95-2220-E-007-037, and Ministry of Economic Affairs under Grant No. MOEA-95-EC-17-A-01-S1-031.

ICCAD'06, November 5-9, 2006, San Jose, CA

distortion of the vias in that area will become serious and hence the yield/reliability of the design will become worse. Therefore, after inserting redundant vias into a design, the maximum via density rule should be re-verified.



(a) Before line end extension



(b) After line end extension

Fig. 2. Via pattern distortions.

Another typical via related design rule is the via extension rule, which demands that, for a via, the portion of the connected metals must be extended beyond the edges of the via cut by at least E μm (see Fig. 2(a)), where E is a process-dependent constant. With the extension distance, the slight cut misalignment during manufacturing can be tolerated. However, as the feature size of a layout object goes below the wavelength of the light used by optical lithography equipment, pattern distortions, such as line-end shortening, increase and aggravate the cut misalignment problem. As shown in Fig. 2(a), we can see that after line-end shortening, there is rare spacing to tolerate the cut misalignment phenomenon. Extending the line-ends of metals for a via (i.e., line end extension) is another method for improving the via yield. Line end extension[1] is to broaden the extension distance and thus can ease the cut misalignment problem caused by line-end shortening, as shown in Fig. 2(b).

In this paper, we study three via yield/reliability improvement problems in the post-routing stage. The first one is the redundant via insertion problem. For this problem, we accelerate the MIS-based approach proposed in [13] to solve it. The approach in [13] consists of the step of conflict graph construction followed by the step of solving an MIS problem on the graph. We present two methods to reduce the run time of the MIS-based approach by speeding up the conflict graph construction step and reducing graph size to facilitate the computation of an MIS solution, respectively. The experimental results indicate that the accelerated MIS-based approach is up to 3X faster without hurting solution quality. The second problem we consider in this paper is simultaneous redundant via insertion and line end extension. We formulate the problem as a maximum weighted independent set (MWIS) problem and enhance the accelerated MIS-based approach (which is originally designed only for the unweighted version) to solve it. Both steps of the accelerated MIS-based approach are modified to consider line-end extended vias as well. The experimental results indicate that the total number of inserted redundant vias and line-end extended vias is very close to the upper bound in each test case. The third problem is to simultaneously consider redundant via insertion and line end extension subject to the maximum via density rule, and a two-stage approach is presented to solve it. In the first stage, by ignoring the maximum via density rule, we enhance the MWIS-based approach to insert redundant vias and extend line ends as much as possible, and at the same time find the set of regions which violate the maximum via density rule. In the second stage, we remove redundant vias from those violating regions such that after the removal, the maximum via density rule is met while the total amount of redundant vias removed is minimized. This density-aware redundant via removal problem is formulated as a set of zero-one integer linear programming (0-1 ILP) problems each of which can be solved independently without sacrificing the optimality. The experimental results indicate that our 0-1 ILP approach also runs efficiently.

To the best of our knowledge, the second and third problems have not been addressed before in the literature. Although an elegant approach based on the MIS formulation was recently proposed for the first problem [13], we are still able to incorporate novel speed-up methods into it.

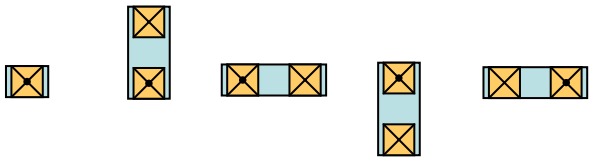
The rest of this paper is organized as follows. In section II, we review the MIS-based approach given in [13] , and then describe two methods to improve its efficiency in section III. In section IV, we detail how to formulate the problem of simultaneous redundant via insertion and line end extension as a MWIS problem, and how to modify the accelerated MIS-based approach to solve the problem. In section V, the two-stage approach for solving the problem of simultaneous redundant via insertion and line end extension subject to the maximum via density rule is described. Section VI reports experimental results, and we conclude the paper in section VII.

II. Redundant Via Insertion

In this section, we review the problem formulation of post-routing redundant via insertion and the MIS-based approach given in [13]. Most of the notation and definitions are from [13].

A. Redundant Via

The manufacturing technology is assumed to consist 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} and its position is specified by its center. We also assume that a set of via related design rules is given, and SP is the spacing between two metals or via cuts[2]. We will use the symbols $X_{LL}(B)$ and $X_{UR}(B)$ ($Y_{LL}(B)$ and $Y_{UR}(B)$, respectively) to represent the x-coordinates (y-coordinates, respectively) of the lower left and upper right corners of the bounding box[3] of a layout object B, respectively.



(a) Single via (b) DVU (c) DVR (d) DVD (e) DVL

Fig. 3. Double via types.

A single via together with a redundant via inserted next to it is defined as a *double via*, and according to the position of a redundant via, a double via can be categorized into four types, as shown in Fig. 3[4] . (Note that in Fig. 3, each square with the × symbol inside is called the via cut for a via.) Given a single via i, its double via of type j ($j \in \{DVU, DVD, DVL, DVR\}$) is denoted by $dv(i,j)$. Besides, a double via (or its corresponding redundant via) is said to be *feasible* if replacing the single via with it will not violate any design rule (excluding the maximum via density rule to be defined in section V), assuming none of

[1] Because a fat via [4] will induce more capacitance and take more area than a line-end extended via, we choose to consider line-end extended vias in this paper.

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

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

[4] The position of the single via is assumed to remain unchanged in each double via pattern. It should be mentioned that all our approaches presented in this paper can be easily extended to consider other double via patterns where the position of the single via is changed [4].

the other single via has any redundant via inserted in the design. The problem of post-routing redundant via insertion is reduced to a maximum independent set problem (see Definition 1 and Problem 1) in [13].

Definition 1. *(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 dv(i',j') will cause design rule violations when both exist in the design.

Problem 1*. 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).*

To solve Problem 1, the MIS-based approach proposed in [13] consists of the step of conflict graph construction and the step of finding an MIS solution. In the next two subsections, we review the algorithm for conflict graph construction and the heuristic for finding an MIS solution, respectively.

B. Conflict Graph Construction

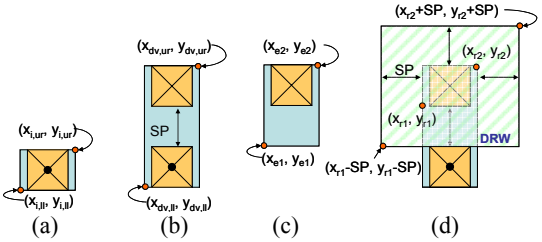
The conflict graph construction algorithm, called *GCA*, constructs the vertex set and edge set of a conflict graph simultaneously. The following definitions on *DVE* and *DRW* are from [13] and required for explaining *GCA*.

Definition 2. *(DVE)*

Suppose the bounding box of a single via i is $R_i=[x_{i,ll}, x_{i,ur}] \times [y_{i,ll}, y_{i,ur}]$ (see Fig. 4 (a)) and the bounding box of a double via dv(i,j) is $R_{dv(i,j)}=[x_{dv,ll},x_{dv,ur}] \times [y_{dv,ll}, y_{dv,ur}]$ (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 3. *(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.)



(a) (b) (c) (d)

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

GCA first sorts all single vias by their *x*-coordinates in the non-decreasing order, and constructs an R-tree [2] for each metal layer to store the bounding boxes of all the layout objects on that layer. (Note that the bounding boxes of all single vias will be also stored in these R-trees) According to the sorted via sequence, denoted 1, 2, …, *n*, each single via will be processed orderly as follows.

Suppose the single via being under consideration is *i*, where $1 \le i \le n$. For each double via *dv*(*i*,*j*), where $j \in \{DVU, DVD, DVL, DVR\}$, *DRW*(*i*,*j*) is used as the query window to do intersection range query on the R-trees of adjacent metal layers when checking if *dv*(*i*,*j*) is feasible. If *dv*(*i*,*j*) is feasible, the corresponding vertex $v_{i,j}$ will be added to the conflict graph.

To efficiently construct the edges in the conflict graph, *GCA* maintains a dynamically updated R-tree, called *VNC* (which is empty at the beginning), to store the *DVE*'s for those feasible double vias which have been identified. Right before checking if each *dv*(*i*,*j*) is feasible or not, for each element of *VNC*, if its right boundary is to the left of the left boundary of *DRW*(*i*,*DVL*), it is impossible to overlap with any *DRW*(*i'*,*j'*) for all *i'*, *j'*, with $i \le i' \le n$ and $j' \in \{DVU, DVD, DVL, DVR\}$, and therefore it will be deleted from *VNC*.

If *dv*(*i*,*j*) is identified being feasible, *DRW*(*i*, *j*) is again used as the query window to do intersection range query on *VNC*. For each *DVE*(*i'*, *j'*) in *VNC*, if it intersects with *DRW*(*i*, *j*), the edge $(v_{i,j}, v_{i',j'})$ will be added to the conflict graph. Finally, since a single via can only be replaced with one double via, *GCA* creates an edge for each pair of vertices each of which corresponds to a feasible double via of *i*. For each feasible double via *dv*(*i*,*j*), its corresponding *DVE*(*i*,*j*) is inserted to *VNC*.

C. Heuristic for Finding an MIS Solution

[13] presents a heuristic, called *H2K*, to find an MIS solution from a conflict graph. First, *H2K* uses the feasible number and degree of a vertex as the first and second keys to construct a priority queue which stores all the vertices of a conflict graph. The feasible number of a vertex $v_{i,j}$ is defined to be the number of vertices $v_{i',j'}$'s in the conflict graph such that $i=i'$ and $j \neq j'$ (i.e., the number of the other feasible double vias originating from the same single via). A vertex has higher priority in the priority queue if it has smaller feasible number and degree.

Then *H2K* finds an MIS solution in an iterative manner. At each iteration, a vertex subset of pre-defined size *k* is extracted from the priority queue, and the subgraph induced by the vertex subset is obtained. Then, a maximal independent set solution on the subgraph is found (by any existing MIS solver) and added to the final solution. Finally, the conflict graph and priority queue are updated by removing those vertices appearing in the maximal independent set and their adjacent vertices and incident edges. Note that the feasible number of a vertex might become decreased in the updated conflict graph. *H2K* will terminate when the conflict graph or priority queue has no remaining vertices.

III. Methods for Speeding up the MIS-Based Approach

In this section, we present two methods to speed up the MIS-based approach. One is to accelerate the construction of a conflict graph and the other is to reduce the size of a conflict graph to which *H2K* will be applied. They are detailed in the following two subsections.

A. Speed-up Method for Conflict Graph Construction

GCA constructs the R-tree of each metal layer statically right at the beginning to store the bounding boxes of all the original layout objects on the layer, and uses the R-trees of two adjacent metal layers for checking if a double via is feasible. Since a double via may induce design rule violations to a layout object only if they both locate in nearby grids, keeping in the R-trees the layout objects which are far enough from the single via being under consideration is not necessary. This implies that the run time of range queries on the R-trees has room to improve. In this subsection, we present a method to speed up *GCA* by dynamically maintaining an R-tree for each metal layer. We make the following modifications to *GCA*.

The R-tree of each metal layer is initially empty. For each metal layer, all its layout objects are sorted by the *x*-coordinates of the lower left corners of their bounding boxes in the non-decreasing order. Suppose via *i* located at (x_i, y_i) is the single via being under consideration. If none of the *x*-coordinates of the single vias that have been processed is equal to x_i, each R-tree will get updated as follows. Suppose $X_{LL}(dv(i,DVL))$ and $X_{UR}(dv(i,DVR))$ are equal to x_{ll} and x_{ur}, respectively, as shown in Fig. 5. We first delete from each R-tree all the elements

contained in the range $[-\infty, x_{ll}-SP]\times[-\infty, +\infty]$, such as O_1 shown in Fig. 5, because they will not induce any design rule violation to all double vias of each single via which has not been processed by *GCA* yet. Then for each layout object OBJ_R remaining in the sorted order, if $X_{LL}(OBJ_R)$ is less than $x_{ll}-SP$, it is removed from the sorted order; on the other hand, if $X_{UR}(OBJ_R)$ is within the range $[x_{ll}-SP, x_{ur}+SP]$, it is removed from the sorted order and inserted into its corresponding R-tree. For each metal layer, this process is repeated until the first layout object OBJ_F with $X_{LL}(OBJ_F)$ greater than $x_{ur}+SP$ (such as O_2 shown in Fig. 5) is reached or no object remains in the sorted sequence.

With the above modifications, we can perform *GCA* to construct the vertex and edge sets of a conflict graph without keeping the whole layout objects of a design in the R-trees at all time. With possibly less layout objects stored in the R-trees, the range query time could be reduced. We name the modified conflict graph construction algorithm as *IGCA*.

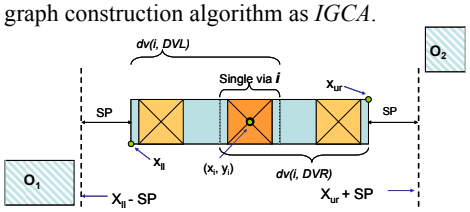


Fig. 5. The geometric information around a single via.

Fig. 6 illustrates how *IGCA* works. In Fig. 6(a), there is a layout consisting of seven wire segments, $O_1, O_2 \ldots, O_7$, and four vias, $V_1, V_2, \ldots, V_4$. Suppose the single via being under consideration is V_3, and the layout objects O_1, O_2 and V_1 will not induce any design rule violation to any double via of V_3 or V_4. For *GCA*, the R-trees for metal layers m and m+1 will consist of all layout objects on the corresponding layers, as shown in Fig. 6(b). However, the layout objects O_1, O_2, and V_1 are unnecessary when checking if a double via of V_3 is feasible; hence, the amount of layout objects stored in the R-trees for metal layers m and m+1 is reduced by *IGCA*, as shown in Fig. 6(c).

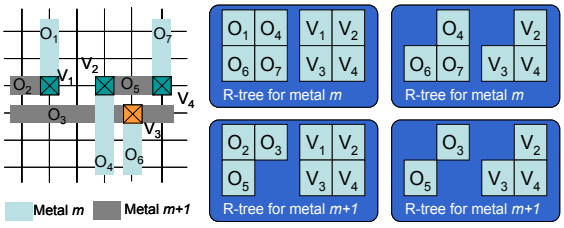


Fig. 6. Illustration for *ICGA*.

B. Graph Reduction

In this subsection, we present a graph reduction algorithm, called *GRA*, which selects and adds vertices into the final MIS solution during the construction of a conflict graph; as a result, the size of the conflict graph is reduced before *H2K* is applied. Before describing the details of *GRA*, we need to define what internal and external edges are.

Definition 4. *(Internal and external edges)*

Given a conflict graph G(V,E), an edge e $(v_{i,j}, v_{i',j'}) \in E$ *is said to be internal if and only if* $i=i'$ *and* $j \neq j'$*, i.e.,* $v_{i,j}$ *and* $v_{i',j'}$ *correspond to two different double vias of the same single via. An edge is said to be external if it is not internal.*

Suppose the single via i located at (x_i, y_i) is being under consideration by *IGCA* and none of the x-coordinates of the single vias that have been processed is equal to x_i. For each single via i' that has been processed by *IGCA*, if $X_{UR}(dv(i',DVR))$ is less than or equal to $X_{LL}(dv(i,DVL))-SP$, and there is one vertex $v_{i',j}$ (corresponding to a feasible double via of the single via i') which is in the current conflict graph and has no external edge, then *GRA* will select $v_{i',j}$ and add it to the final MIS solution. Since there will never be an external edge incident to $v_{i',j}$, inserting the double via $dv(i',j)$ will not cause any design rule violation or prevent any possible insertion of other doubles vias originating from single vias other than i'. Finally *GRA* deletes all adjacent vertices of $v_{i',j}$ and all edges incident to those deleted vertices from the current conflict graph. As a result, *H2K* can solve the MIS problem on the reduced conflict graph without hurting the quality of the MIS solution.

Fig. 7 illustrates how *GRA* works. In Fig. 7(a), there are four single vias in the design, and they are numbered to form the sorted sequence. Suppose the single via 3 is being considered by *IGCA* and its x-coordinate is different from the x-coordinates of single vias 1 and 2. Besides, we also assume that $X_{UR}(dv(1,DVR))$ is less than or equal to $X_{LL}(dv(3,DVL))-SP$. The conflict graph right before adding the vertices corresponding to the feasible double vias of single via 3 is shown in Fig. 7(b), in which the bold edge connecting $V_{1,DVR}$ and $V_{2,DVL}$ stands for an external edge. Because there will be no new edge to be added for connecting any vertex corresponding to a feasible double via of single via 1, either $V_{1,DVU}$ or $V_{1,DVL}$ (but not both) can be selected as an element of the final MIS solution immediately.

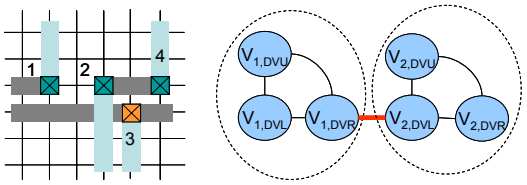


Fig. 7. Illustration for how *GRA* works.

C. Overall Approach

The accelerated MIS-based approach, which first uses *IGCA* and *GRA* to construct and reduce the conflict graph from a routed design, and then applies *H2K* on the conflict graph, is named *IMBA*.

IV. Simultaneous Redundant Via Insertion and Line End Extension

In this section, we first define the problem of simultaneous redundant via insertion and line end extension, and reduce it into a MWIS problem. We then describe how to modify the accelerated MIS-based approach, i.e., *IMBA*, to solve the problem.

A. Line-end Extended Via

We assume that the extension distance for a line-end extended via is αE, where E is the extension distance specified in the via extension rule for a single via, and α is a process-dependent constant great than one. The structure of a line-end extended via is illustrated in Fig. 8. We use *LE* to represent the line-end extended via structure.

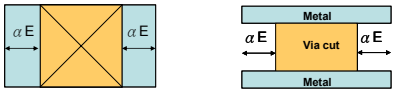


Fig. 8. The structure of a line-end extended via.

B. Problem Formulation

The problem of simultaneous redundant via insertion and line end extension is defined as follows.

Problem 2. *Given a detailed routing solution, without re-routing any signal net, the problem asks to replace single vias on signal nets with double vias or line-end extended vias such that after replacement, both the amounts of double vias and line-end extended vias are as large as possible. In addition, two conditions must be satisfied after replacement: First, each single via either remains unchanged, or is replaced by a double*

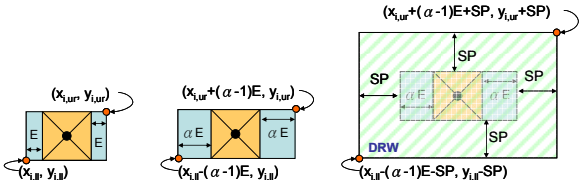
via or a line-end extended via. Second, no design rule is violated.

Extending the line end of a single via increases the manufacturability, but it also increases the via capacitance. On the other hand, although adding a redundant via next to a single via also increases the capacitance, it reduces the via resistance as well. Besides, redundant vias can improve not only yield but also reliability of a design. Therefore, adding a redundant via adjacent to a single via will has a higher priority than extending the line end of the single via in Problem 2.

In fact, we can view the line-end extended via structure *LE* as a "pseudo" double via type. Given a single via *i*, let *dv*(*i, LE*) denote its line-end extended via; *dv*(*i,LE*) is said to be *feasible* if replacing *i* with *dv*(*i,LE*) will not violate any design rule, under the assumption that the other single vias remain unchanged. Given a single via *i*, both *DRW*(*i,LE*) and *DVE*(*i,LE*) are defined below.

Definition 5. *(DVE and DRW for a line-end extended via)*

Suppose the bounding box of a single via i is $R_i=[x_{i,ll}, x_{i,ur}] \times [y_{i,ll}, y_{i,ur}]$*(see Fig. 9(a)). The DVE(i, LE) and DRW(i, LE) are defined as* $[x_{i,ll}-(\alpha-1)E, x_{i,ur}+(\alpha-1)E] \times [y_{i,ll}, y_{i,ur}]$ *(see Fig. 9(b)) and* $[x_{i,ll}-(\alpha-1)E-SP, x_{i,ur}+(\alpha-1)E+SP] \times [y_{i,ll}-SP, y_{i,ur}+SP]$ *(see Fig. 9 (c)), respectively.*



(a) a single via (b) *DVE* for *LE* (c) *DRW* for *LE*

Fig. 9. Illustration of *DVE* and *DRW* for *LE*.

The definition of a conflict graph is modified below to account for a *weighted* conflict graph which captures line-end extended vias as well.

Definition 6. *(Weighted conflict graph)*

A weighted conflict graph G(V,E) is an undirected vertex-weighted graph constructed from a detailed routing solution. For each single via i on a signal net, if its double via of type j ($j \in \{DVL, DVR, DVU, DVD, LE\}$*) is feasible, there exists a vertex* $v_{i,j}$ *in V. An edge* $(v_{i,j}, v_{i',j'}) \in E$ *if i=i', or dv(i,j) and dv(i',j') will cause design rule violations when both exist in the design. For each* $v_{i,j}$ *in V, if it corresponds to a line-end extended via, its weight is set to 1; otherwise its weight is set to n+1. Here n is the number of vertices in V each of which corresponds to a line-end extended via.*

Given a weighted conflict graph, the MWIS problem is defined as follows. For each independent set of the graph, its cost is measured by the sum of the weights of all the vertices in the set. The MWIS problem asks to find an independent set with largest cost. It is not hard to prove that Problem 2 can be reduced to the MWIS problem. In the following subsection, we will describe how to enhance the accelerated MIS-based approach, i.e., *IMBA*, which combines *ICGA*, *GRA*, and *H2K*, to construct a weighted conflict graph and find an MWIS solution. We name this modified approach as *EMBA*.

C. MWIS-based Approach

Given a routed design, the corresponding weighted conflict graph *G*(*V*, *E*) is constructed by *ICGA* and *GRA* with the following modifications. *ICGA* is modified such that it will treat each line-end extended via as a double via, and assign a weight to each vertex. This modified *IGCA* is named *WIGCA*. Since maximizing the amount of inserted redundant vias is the most important goal, *GRA* is modified in such a way that none of the vertices corresponding to feasible line-end extended vias is considered as a candidate for adding to the final independent set solution even if no external edge is incident to the vertex. This modified *GRA* is called *WGRA*. Let $MWIS_1$ be the set of vertices selected and added to the final solution by *WGRA*. The set $MWIS_1$ contains redundant vias only.

After *G* is constructed, *H2K* is applied to find a maximal weighted independent set $MWIS_2$ of *G* with the following modifications. First, each vertex in the priority queue is added with the third key. If a vertex corresponds to a line-end extended via, it will have a lower priority on this key. With this modification, for vertices having the same feasible number and degree, the ones corresponding to redundant vias will be extracted first, and hence have higher chances to be included in the final solution. Second, a maximal weighted independent set is found from the extracted subgraph at each iteration. This modified *H2K* is called *WH3K*.

Finally, the union of $MWIS_1$ and $MWIS_2$ is output as the final solution.

V. Via-Density-Constrained Simultaneous Redundant Via Insertion and Line End Extension

In this section we first describe the via density rules and give the problem formulation of simultaneous redundant via insertion and line end extension under the maximum via density rule. We then detail our two-stage approach for solving the problem.

A. Via Density Rules and Problem Formulation

To analyze via density, each via layer VIA_i is partitioned into a set *R*(*i*) of overlapping rectangular regions each of which has the same width *W* and height *H*, where *W* and *H* are process-dependent constants. All the regions in *R*(*i*) are organized into an *m*-row by *n*-column structure. See Fig. 10 for an illustration. We use *r*(*i,j,k*) to represent the region which is in *R*(*i*) and located at row *j* and column *k*. For any two neighboring regions *r*(*i,j,k*) and *r*(*i,j,k*+1) in the same row, such as regions *A* and *B* in Fig. 10, their overlapped distance in the *x*-direction is defined to be $(1/\lambda)W$, where λ is a process-dependent constant. Similarly, for any two neighboring regions *r*(*i,j,k*) and *r*(*i,j*+1,*k*) in the same column, such as regions *A* and *C* in Fig. 10, their overlapped distance in the *y*-direction is defined to be $(1/\beta)H$, where β is a process-dependent constant. Therefore, a via is possible to be located in more than one region.

For each region *r*(*i,j,k*), its via density, denoted by *density*(*i,j,k*), is defined as the number of vias[5] located in it. Two design rules related to via density, called the *minimum via density rule* and the *maximum via density rule*, can be considered. The minimum via density rule requires *density*(*i,j,k*) be greater than or equal to *L* for each region *r*(*i,j,k*), where *L* is a process-dependent constant. On the other hand, the maximum via density rule requires *density*(*i,j,k*) be less than or equal to *U* for each region *r*(*i,j,k*), where *U* is a process-dependent constant.

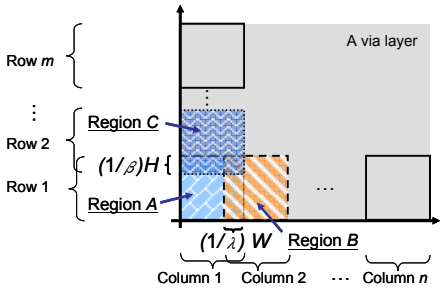


Fig. 10. Illustration for the via regions.

One method to meet the minimum via density rule is to add dummy vias [16] (if necessary), which unlike "normal" vias, are not used to provide signal paths between metal layers. In this paper, we assume the given routed design satisfies both the

[5] Each of such vias can be a single via, a redundant via, or a line-end extended via.

minimum and maximum via density rules, and hence only the maximum via density constraint needs to be taken into account after adding redundant vias. Now Problem 2 is modified below to set Problem 3, while considers the maximum via density rule as well.

Problem 3. *Given a detailed routing solution which already satisfies the minimum and maximum via density rules, without re-routing any signal net, the problem asks to replace single vias on signal nets with double vias or line-end extended vias such that after replacement, both the amounts of double vias and line-end extended vias are as large as possible. In addition, two conditions must be satisfied after replacement: First, each single via either remains unchanged, or is replaced by a double via or a line-end extended via. Second, no design rule, including the maximum via density rule, is violated.*

B. Overview of Our Two-stage Approach

In this subsection, we give an overview of our two-stage approach for solving Problem 3. The details are described in the next two subsections.

In the first stage, the maximum via density rule is ignored, and we modify *EMBA*[6] such that in addition to performing simultaneous redundant via insertion and line end extension, it also reports the "violating" regions each of which violates the maximum via density constraint because of excess redundant vias inserted. Besides, for each violating region, the set of redundant vias that have been inserted in it, and the minimum number of redundant vias that must be deleted from it in order to meet the maximum via density rule are also generated.

In the second stage, we remove from each violating region redundant vias[7] such that after the removal, the maximum via density rule is satisfied while the total amount of removed redundant vias is as small as possible. This redundant via removal problem will be formulated as a set of zero-one integer linear programming (0-1 ILP) problems each of which is solved independently without sacrificing the optimality.

In the next two subsections, we first describe how to modify *EMBA* so as to efficiently find all the violating regions; we then explain the details of the 0-1 ILP approach.

C. Via Density Calculation

In this subsection, we describe how to extend *EMBA* to generate the information on each violating region, in addition to performing simultaneous redundant via insertion and line end extension.

For each region $r(i,j,k)$, let $\#sv(i,j,k)$ denote the number of single vias located in the region, and let $sv\_set(i,j,k)$ denote the set of single vias, each of which has at least one feasible redundant via (which may or may not be inserted into the design later) located in the region. It is not hard to see that a region $r(i,j,k)$ will not violate the maximum via density rule after adding redundant vias, if $\#sv(i,j,k)+|sv\_set(i,j,k)|$ is less than or equal to U. Let $DANGER=\{r(i,j,k) \mid \#sv(i,j,k)+|sv\_set(i,j,k)| > U$, for all i, j and $k\}$ be the set of regions which are possible to violate the maximum via density rule after redundant via insertion. For each feasible redundant via v', let $reg\_set(v')$ be the set of regions each of which is in *DANGER* and contains v'. During the course of constructing the weighted conflict graph by *WIGCA*, we want to calculate $\#sv(i,j,k)$, $sv\_set(i,j,k)$, *DANGER* and $reg\_set(v')$ as well. Therefore, we add the following extensions to *WIGCA*.

At the beginning, $\#sv(i,j,k)$ is 0, and $sv\_set(i,j,k)$, *DANGER* and $reg\_set(v')$ are all empty sets. For each via layer VIA_i, all its regions are sorted by the x-coordinates of their lower left corners in the non-decreasing order, and an R-tree is dynamically maintained to store its regions. These R-trees are initially empty. Suppose v located at (x_v, y_v) of the via layer VIA_i is the single via being under consideration by *WIGCA*. If none of the x-coordinates of the single vias that have been processed is equal to x_v, the R-tree for each via layer will be updated as follows. Suppose $X_{LL}(dv(v,DVL))$ and $X_{UR}(dv(v,DVR))$ are equal to x_{ll} and x_{ur}, respectively, as shown in Fig. 11. We first extract and delete from each R-tree all the regions contained in the range $[-\infty, x_{ll}]\times[-\infty, +\infty]$ (such as Region 1 shown in Fig. 11), because none of the single vias which have not been processed by *WIGCA*, and none of their feasible double vias are located in these regions. For each deleted region $r(i,j,k)$, if $\#sv(i,j,k)+|sv\_set(i,j,k)| > U$, we add $r(i,j,k)$ to both *DANGER* and $reg\_set(v')$ for each feasible redundant via v' that is contained in $r(i,j,k)$. Then for each region $r(i,j,k)$ remaining in the sorted order, if $X_{LL}(r(i,j,k))$ is less than x_{ur}, it is removed from the sorted order and inserted into the corresponding R-tree. For each via layer, this updating process is repeated on its R-tree until the first region $r(i',j',k')$ with $X_{LL}(r(i',j',k'))$ greater than x_{ur} (such as Region 2 shown in Fig. 11) is reached or no object remains in the sorted order.

After updating the R-trees (if necessary), we use the via cut of v as the query window to do range query on the R-tree corresponding to the via layer VIA_i on which v is located. For each queried region $r(i,j,k)$ that encloses the query window, we increase $\#sv(i,j,k)$ by one. Besides, for each feasible redundant via v' of v, we use its via cut as the query window to do range query on the corresponding R-tree as well; for each queried region $r(i,j,k)$ that encloses the query window, v' is found to be contained in $r(i,j,k)$, and if v is not in $sv\_set(i,j,k)$ yet, we add v to $sv\_set(i,j,k)$.

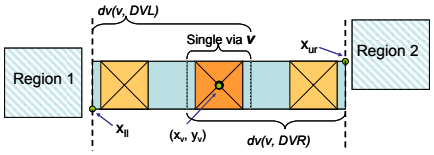


Fig. 11. Illustration for the dynamically updating strategy.

After all single vias of the design have been processed by *WIGCA*, for any remaining region $r(i,j,k)$ in each R-tree, if $\#sv(i,j,k)+|sv\_set(i,j,k)| > U$, we add $r(i,j,k)$ to both *DANGER* and $reg\_set(v')$ for each feasible redundant via v' that is contained in $r(i,j,k)$.

After applying *WH3K* to perform simultaneous redundant via insertion and line end extension, let R_{vio} denote the set of all regions each of which violates the maximum via density rule; besides, for each region $r(i,j,k)$ in R_{vio}, let $rv\_set(i,j,k)$ denote the set of all redundant vias that have been inserted into $r(i,j,k)$, and let $\#del(i,j,k)$ be the minimum number of redundant vias that must be deleted from $r(i,j,k)$ in order to meet the maximum via density rule (i.e., $density(i,j,k) - \#del(i,j,k) = U$). R_{vio}, $rv\_set(i,j,k)$ and $\#del(i,j,k)$ are computed as follows. For each redundant via v' that has been inserted into the design, we add v' to $rv\_set(i,j,k)$ for each $r(i,j,k) \in reg\_set(v')$. For each region $r(i,j,k) \in DANGER$, if $\#sv(i,j,k)+|rv\_set(i,j,k)| > U$, it will be added to R_{vio}, and $\#del(i,j,k)$ will be set to $\#sv(i,j,k)+|rv\_set(i,j,k)| - U$.

D. Redundant Via Removal

When the first stage of our approach is done, R_{vio}, $rv\_set(i,j,k)$ and $\#del(i,j,k)$ are ready for use in the second stage, for all $r(i,j,k) \in R_{vio}$. If R_{vio} is empty, it means that the solution of simultaneous redundant via insertion and end line extension generated in the first stage already satisfies the maximum via density rule, and therefore nothing is done in the second stage. On the other hand, if R_{vio} is not empty, the following problem is solved in the second stage.

[6] We can also discuss a variant of Problem 3, in which only redundant via insertion is considered. For this variant, we can modify *IMBA* and apply it in the first stage.

[7] Line-end extended vias will not be candidates for removal because each of them corresponds to a single via but with larger extension distance and therefore removing it will make a net become disconnected.

Problem 4. *Given R_{vio}, rv_set(i,j,k) and #del(i,j,k) for each r(i,j,k)* $\in R_{vio}$*, the problem asks to remove redundant vias from each rv_set(i,j,k) at least by the amount #del(i,j,k) such that after the removal, each resulting region r(i,j,k) satisfies the maximum via density rule while the total number of redundant vias removed is minimized.*

We now describe a 0-1 ILP-based method to optimally solve Problem 4. Let $VEX = \bigcup_{r(i,j,k) \in R_{vio}} rv\_set(i,j,k)$, be the set of redundant vias, say rv_1, rv_2, …, $rv_{|VEX|}$, to be considered for removal. Problem 4 is formulated as a 0-1 ILP problem below.

Minimize $\sum_{rv_p \in VEX} X_p$

s.t.

$\sum_{rv_p \in rv\_set(i,j,k)} X_p \geq \#del(i,j,k), \forall r(i,j,k) \in R_{vio}$

$X_p \in \{0,1\}, \forall rv_p \in VEX$

where each redundant via rv_p is associated with a zero-one variable X_p and if X_p is 1, the redundant via rv_p will be deleted from the design. The number of constraints and variables in the 0-1 ILP problem are equal to the number of regions violating the maximum via density rule, and the number of redundant vias located in these regions, respectively. For a large and/or high routing congestion design, the number of variables and/or constraints of the 0-1 ILP problem could become large.

In order to reduce the time complexity, we partition R_{vio} into disjoint subsets Sub_1, Sub_2, …, Sub_t (with $t \leq |R_{vio}|$) such that the following two conditions hold for each subset Sub_q : (1) If there are two or more regions in Sub_q, then for each region $r(i,j,k) \in Sub_q$, there must exist a redundant via rv and another region $r(i',j',k') \in Sub_q$ such that rv is contained in both regions $r(i,j,k)$ and $r(i',j',k')$. (2) For each region $r(i,j,k) \in Sub_q$, if a redundant via rv is contained in $r(i,j,k)$, then rv is not contained in any region in $R_{vio} - Sub_q$. It is not hard to verify that we can individually formulate and solve a 0-1 ILP problem for each Sub_q without sacrificing the optimality. Besides, for each Sub_q, if it has only one region, we will arbitrarily delete the redundant vias from the region by the amount $\#del(i,j,k)$. The partitioning of R_{vio} can be correctly done by the Union-Find algorithm [3].

VI. Experimental Results

We used a 0.18μm technology which has 5 metal layers in our experiments. For simplicity we directly used the R*-tree package [11] for indexing 2-dimensional geometric information of layout objects. We used the qualex-ms [9] as the MIS or MWIS solver. Besides, we also limited the subgraph extracted at each iteration of *H2K* or *MH3K* to consist of 1500 vertices at most. Moreover, we used the lp_solver [8] as our 0-1 ILP solver. All the experiments were conducted on a Linux based machine with 2.4G processor and 4GB memory. All the run times are measured in seconds.

Table 1: The test cases

Case	Size(μm)	#Nets	#I/Os	#Vias	#Layers	#Objects
C1	350.000*350.000	4309	20	24594	5	218215
C2	419.433*413.28	5252	211	41157	5	268669
C3	799.124*776.16	18157	85	127059	5	933852
C4	691.272*680.400	17692	415	151912	5	934073
C5	1383.482*1375.92	44720	99	357386	5	2851612

The test cases we used are the same as those in [13] and the detailed information is shown in Table 1; for each test case, the first column shows the circuit name, "Size(μm)" gives the layout dimension, "#Nets" shows the number of nets, "#I/Os" gives the number of I/O pins, "#Vias" shows the total number of single vias, and "#Layers" gives the number of metal layers used. Finally, "#Objects" gives the total number of layout objects including pins, vias, blockages and wire segments.

To see how much run time reduction could be achieved by our two speed-up methods, *IGCA* and *GRA*, we first focus on the problem of redundant via insertion. The results are shown in Table 2, where the columns "[13]", "IGCA+H2K" and "IMBA" are the redundant via insertion results after applying *H2K* on the conflict graphs generated by *GCA*, *IGCA*, and *IGCA+GRA*, respectively. The column under "#RV" gives the number of redundant vias inserted by the approach given in [13], and the column "#diff" under each of "IGCA+H2K" and "IMBA" gives the difference between the number of redundant vias inserted by [13] and the number of redundant vias inserted by "IGCA+H2K" or "IMBA". Each column "T(s)" gives the run time of each approach, and the column "Speed-Up" under "IGCA+H2K" or "IMBA" gives the speed-up ratio of "IGCA+H2K" or "IMBA" over "[13]".

From Table 2, we can see that running *IGCA* alone is 1.98~2.46X (2.04X on average) faster than [13] while maintaining the same amount of redundant vias inserted for each test case. When we applied both *IGCA* and *GRA*, the speed-up ratio becomes even higher, ranging from 2.52 to 3; the average speed-up ratio is 2.92. Except for the test case C4, the total number of redundant vias inserted by *IMBA* for any other case is the same as that reported by [13]. The reason why *IMBA* inserted 2 less redundant vias for the test case C4 is that the size of the conflict graph was reduced before *H2K* was applied, and *H2K* could only guarantee to generate a maximal independent set (which is not necessarily a maximum independent set).

Table 2: Experimental results of [13], *IGCA+H2K* and *IMBA*

	[13]		IGCA+H2K			IMBA		
	#RV	**T(s)**	**#diff**	**T(s)**	**Speed-Up**	**#diff**	**T(s)**	**Speed-Up**
C1	17461	32	0	13	2.46X	0	11	2.90X
C2	28507	43	0	20	2.15X	0	17	2.52X
C3	91461	192	0	86	2.23X	0	67	2.86X
C4	101765	203	0	98	2.07X	-2	72	2.81X
C5	254428	710	0	361	1.98X	0	236	3.00X
Avg.		**1**			**2.04X**			**2.92X**

For the problem of simultaneous redundant via insertion and line end extension, there is no existing approach. Therefore we develop a two-stage MIS-based approach called *2SMIS*, and compare it with our approach *EMBA*. The main idea of *2SMIS* is to insert redundant vias in the first stage, and then perform line end extension in the second stage. *2SMIS* was implemented based on an extension of the MIS-based approach [13] by applying *H2K* twice, one on a conflict graph consisting of vertices corresponding to redundant vias only, and the other on another conflict graph consisting of vertices corresponding to line-end extended vias only. Note that the number of redundant vias inserted by *2SMIS* is always the same as that produced by the MIS-based approach [13].

In Table 3, the rows "2SMIS" and "EMBA" show the experimental results of the two approaches *2SMIS* and *EMBA*, respectively. The column "Upp." denotes the number of single vias each of which has at least one feasible double via or one feasible line-end extended via. (Note that "Upp" can be thought of as an upper bound on the total number of redundant vias and line-end extended vias that can be inserted.) "#RV" and "#LE" show the numbers of redundant vias and line-end extended vias inserted, respectively. "R(%)" gives the ratio of the sum of "#RV" and "#LE" (i.e., "#RV" + "#LE") to "Upp.". "T(s)" gives the CPU time of each approach. "Speed-Up" gives the speed-up ratio of "EMBA" over "2SMIS".

From Table 3, we can see that compared to *2SMIS*, our approach *EMBA* is able to insert the same (for three test cases) or almost the same (differs by 1 or 2 for the other two test cases) number of redundant vias, and more line-end extended vias (for all test cases). (Note that *EMBA* could insert less redundant vias because it is just a heuristic for the MWIS problem.) As a result,

the total amount of redundant vias and line-end extended vias inserted by our approach is closer to the upper bound (the ratio ranges from 99.47% to 99.74%) than *2SMIS*. Besides, *EMBA* was 2.53~3.11X (2.92X on average) faster than *2SMIS*.

Table 3: Experimental results of *EMBA* and *2SMIS*

Case	Upp.		#RV	#LE	R(%)	T(s)	Speed -Up
C1	18258	**2SMIS**	17461	741	99.69	33	2.53X
		EMBA	17461	751	99.74	13	
C2	29106	**2SMIS**	28507	513	99.70	46	2.42X
		EMBA	28506	519	99.72	19	
C3	93675	**2SMIS**	91461	1946	99.71	206	2.74X
		EMBA	91461	1972	99.74	75	
C4	104263	**2SMIS**	101765	1917	99.44	217	2.64X
		EMBA	101765	1947	99.47	82	
C5	261035	**2SMIS**	254428	5730	99.66	814	3.11X
		EMBA	254426	5791	99.68	261	
Avg.							**2.92X**

For the problem of simultaneous redundant via insertion and line end extension under the maximum via density rule, the width and height of each region on a via layer were set to 10.08μm and 8.4μm, respectively, for each test case. Both λ and β, defined in section V-A, were set to 3. For implementation simplicity, we used the center of a via to judge whether the via is located in a region. We observed that among all test cases, the maximum number of single vias located in a region was 30, and therefore in order to enforce a very tight maximum via density rule in our experiments, the maximum number of vias, including single vias, redundant vias and line-end extended vias, that could be located in a region was set to 30. The additional run times spent by the first stage of our approach on generating the information on violating regions were 2, 2, 18, 132, and 65 seconds for the five test cases, respectively. In the following discussion, we only focus on the redundant via removal problem.

Table 4. Statistics on the redundant via removal problem

Case	# Regions	#Vio. Regions	#Sub-prob.	#Cand. rv
C1	52500	252	52	2238
C2	74000	412	90	3638
C3	257984	956	223	8478
C4	187200	9363	220	44396
C5	810816	1402	334	12648

The input statistics are shown in Table 4. For each test case, the column "#Regions" denotes the total number of regions. "#Vio. regions" gives the number of regions violating the maximum via density rule, i.e., $|R_{vio}|$. "#Sub-prob." shows the number of subsets which have more than one region, after the partitioning of R_{vio} was done. (Note that "#Sub-prob." also specifies the number of 0-1 ILP problems to be solved.) "#Cand. rv" is the total number of distinct redundant vias that are inserted in the regions of R_{vio}, i.e., $|VEX|$.

We also implemented two heuristics, called GREEDY and RANDOM, for comparative studies. GREEDY first creates a priority queue to store all the redundant vias in *VEX*. Each redundant via is associated with a key which is measured by the number of violating regions containing the redundant via. The method iteratively extracts from the priority queue a redundant via with the largest key, and deletes it from the design. Since deleting a vertex may cause the decrease of the keys for other redundant vias, the priority queue may need update after each deletion. As soon as no violating region exists, the method terminates. The second heuristic, RANDOM, iteratively selects a violating region randomly, and removes excess redundant vias from that region arbitrarily until there exists no violating region.

The number of redundant vias removed by each method and the run time are shown in the corresponding columns under "# of removed rv" and "T(s)" of Table 5, respectively. The columns "0-1 ILP", "Greedy" and "Random" show the results of the 0-1 ILP methods (with and without doing partitioning), GREEDY and RANDOM, respectively. The columns "P" and "NP" under "T(s)" give the run times of both 0-1 ILP methods, respectively, where "P" stands for our method and "NP" is the one without doing partitioning on R_{vio}. It is clear to see that partitioning the whole 0-1 ILP problem into a set of smaller problems helps to improve the run time for each test case; without doing partitioning, it is 6 times slower on average. Surprisingly our 0-1 ILP method with partitioning also runs faster than GREEDY for almost all test cases; on average GREEDY is 3.82 times slower. The reason why GREEDY is slower is mainly due to the overhead caused by updating the priority queue after each via removal. For each test case, although RANDOM runs much faster than our method, the amount of redundant vias removed by it is also much larger. Similarly GREEDY also removed more redundant vias than our method. On average, GREEDY and RANDOM are 1.5% and 29.6% worse in terms of the number of redundant vias removed.

Table 5: The performance comparison

Case	# of removed rv			T(s)			
	0-1 ILP	Greedy	Random	0-1 ILP		Greedy	Random
				P	NP		
C1	387	389	486	0.03	0.09	0.03	<0.01
C2	625	627	748	0.05	0.22	0.08	<0.01
C3	1501	1502	1762	0.13	1.32	0.89	<0.01
C4	10631	10856	14379	14.3	84.0	52.2	0.04
C5	2370	2373	2732	0.26	3.11	3.25	0.01
Avg.	**1**	**1.015**	**1.296**	**1**	**6.00**	**3.82**	**0.003**

VII. Conclusions and Future Works

In this paper, we have studied three post-routing via yield/reliability improvement problems, and have presented novel approaches for them. The experimental results well support all our approaches. Our future work is to study how to consider redundant via insertion, line end extension, and via density rules all together in one single stage.

References

[1] A. B. Kahng, "Research Directions for Coevolution of Rules and Routers", Proc. of ISPD, 2003.

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

[3] Cormen, Leiserson and Rivest, Introduction to Algorithms (Second Edition), The MIT Press, 2001.

[4] F. Luo, Y. Jia and W.-M. Dai, "Yield-Preferred Via Insertion Based on Novel Geotopological Technology", Proc. of ASP-DAC, 2006.

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

[6] G. Xu, Li-Da Huang, D. Z. Pan and M. D. F. Wong, "Redundant-Via Enhanced Maze Routing for Yield Improvement", Proc. of ASP-DAC, 2005.

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

[8] http://lpsolve.sourceforge.net/

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

[10] L. K. Scheffer, "Physical CAD Changes to Incorporate Design for Lithography and Manufacturability", Proc. of ASP-DAC, 2004.

[11] 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.

[12] 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.

[13] K.-Y. Lee and T.-C. Wang, "Post-Routing Redundant Via Insertion for Yield/Reliability Improvement", Proc. of ASP-DAC 2006.

[14] S. Raghvendra and P. Hurat, "DFM: Linking Design and Manufacturing", Proc. of ICVD, 2005.

[15] TSMC Reference Flow 5.0.

[16] V. Pitchumani, B. Landau and J. Brandenburg, Design for Manufacturability : Embedded Tutorial, ASP-DAC 2005.

[17] 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.