An Exact Jumper Insertion Algorithm for Antenna Effect Avoidance/Fixing *

Bor-Yiing Su
Department of Electrical Engineering
National Taiwan University
Taipei 106, Taiwan

Yao-Wen Chang
Graduate Institute of Electronics Engineering
and Department of Electrical Engineering
National Taiwan University
Taipei 106, Taiwan

ABSTRACT

As the process technology enters the nanometer era, reliability has become a major concern in the design and manufacturing of VLSI circuits. In this paper we focus on one reliability issue—jumper insertion in routing trees for avoiding/fixing antenna effect violations at the routing/post-layout stages. We formulate the jumper insertion for antenna avoidance/fixing as a tree-cutting problem. We show that the tree-cutting problem exhibits the properties of optimal substructures and greedy choices. With these properties, we present an $O(V \lg V)$-time exact jumper insertion algorithm that uses the optimum number of jumpers to avoid/fix the antenna violations in a routing tree with V vertices. Experimental results show the superior effectiveness and efficiency of our algorithm.

Categories and Subject Descriptors: J.6 [Computer-Aided Engineering]: Computer-Aided Design

General Terms: Algorithms, Performance

Keywords: Antenna Effect, Jumper

1. INTRODUCTION

As the process technology enters the nanometer era, product reliability and manufacturing yield have become major concerns in the design and manufacturing of VLSI circuits. The fine feature size of modern IC technologies is typically achieved by using plasma-based processes. In nanometer technology, more stringent process requirements cause some advanced high-density plasma reactors adopted in the production lines to achieve fine-line patterns [4]. However, these plasma-based processes will charge conducting components of a fabricated structure. As a result, the accumulated charges may affect the quality of IC's. This is called the *antenna effect*.

During metalization, long floating interconnects act as temporary capacitors and accumulate charges gained from the energy provided by fabrication steps such as plasma etching. A random discharge of the floating node due to subsequent process steps could permanently damage transistors in the IC [5, 6]. For instance, the exposed polysilicon and metal structures connected to a thin-oxide transistor will collect charge from the processing environment (e.g., reactive ion etch) and damage the transistor when the discharging current flows through the thin oxide. The mechanism of antenna damage is not fully understood, but there is experimental evidence indicating when charging occurs and how it may affect the quality of gate oxide [5, 6]. Charging occurs when conductor layers not covered by a shielding layer of oxide are directly exposed to plasma. The amount of such charging is proportional to this plasma-exposed area. If conductor layers are connected to a diffusion layer pattern, such charges are discharged to the substrate through the diffusion; see Figures 1(b), (c), and (d) for illustrations. On the other hand, if the charged conductor layers are connected only to the gate oxide, Fowler-Nordheim (F-N) tunneling current through thin oxide discharges such charges and causes damage to the thin oxide [5]; see Figures 1(b) and (c). As shown in Figure 1, interconnects are manufactured layer by layer. Before a conducting path to the diffusion is formed in metal 2 layer pattern etching (see Figure 1(d)), the interconnects in the poly and metal 1 layers might have accumulated so many charges that they cause damage on the gate in the left of Figure 1(c). (Note that there will not be any antenna violation after a conducting path to the diffusion is formed.)

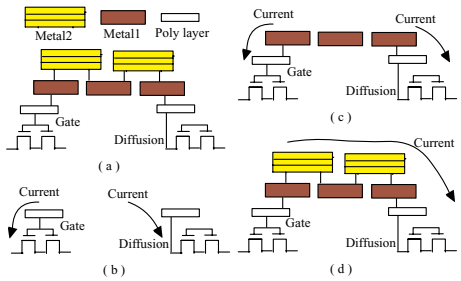

Figure 1: Antenna effect. (a) An example routing. (b) Late stage of poly layer pattern etching of Figure (a). Charge on the left poly pattern is discharged through the gate while charge on the right poly pattern is discharged through the diffusion. (c) Late stage of metal 1 layer pattern etching of Figure (a). Charge on the left metal 1 pattern is discharged through the gate while charge on the right metal 1 pattern is discharged through the diffusion. (d) Late stage of metal 2 layer pattern etching. Charges on all the metal 2 patterns are discharged through the diffusion.

There are three kinds of solutions to reduce the antenna effect [1]:

1. Jumper insertion
2. Embedded protection diode
3. Diode insertion after placement and routing

Comparing the three methods, for method 2 of embedded protection diode, since these diodes are embedded and fixed, they consume unnecessary areas when there is no violation at the connecting wire. In the third method, we need extra space in the chip to place the diodes. Because the number of diodes needed for fixing antenna violations grows dramatically as the feature shrinks, it is hard to preserve enough space for diodes in nanometer IC designs. As a result, jumper insertion becomes the most popular approach for avoiding/fixing antenna violations. The function of jumper insertion can be explained using Figure 2. In Figure 2(a), when the metal 1 layer is manufactured, the gate on the right might be damaged because the large area of the metal 1 interconnection can accumulate sufficient charges to damage the gate. However, if we insert a jumper to route the interconnect on the metal 2 layer as shown in Figure 2(b), the effective

*This work was partially supported by SpringSoft, Inc. and National Science Council of Taiwan under Grant No's. NSC 93-2215-E-002-009, NSC-93-2220-E-002-001, NSC 93-2752-E-002-008-PAE, and NSC 93-2815-C-002-046-E. Emails: b90901130@ntu.edu.tw; ywchang@cc.ee.ntu.edu.tw.

DAC 2005, June 13–17, 2005, Anaheim, California, USA.

conductor layer becomes smaller. Therefore, the stored charge is not enough to damage the gate on the right, and thus we can avoid the antenna violation.

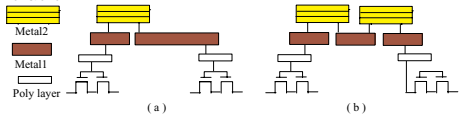

Figure 2: Jumper insertion. (a) Stage before inserting a jumper. (b) Stage after inserting a jumper from the metal 1 layer to the metal 2 layer.

Although jumper insertion is currently the most popular approach for antenna avoidance/fixing, jumpers induce vias that will consume silicon areas and reduce circuit performance. Therefore, it is desired to fix antenna violations by using the least jumpers. Recently, Ho, Chang, and Chen in [3] proposed a bottom-up approach to insert jumpers in a routing tree for antenna avoidance. The work inserts jumpers only beside tree nodes, and its optimality holds only for this special condition of inserting jumpers right beside tree nodes. As an example shown in Figure 3, the wire segment is of $1.3L_{max}$ long, where L_{max} denotes the upper bound for antenna (i.e., any wire longer than L_{max} will violate the antenna rule). For this wire segment, the work in [3] needs two jumpers to fix the antenna violation (see Figure 3(a)) while a single jumper suffices to fix the violation (see Figure 3(b)).

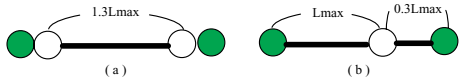

Figure 3: Jumper insertion for a wire of $1.3L_{max}$ long. (a) Two jumpers are needed for fixing the antenna violation if jumpers can be inserted only beside tree nodes, as the assumption made in [3]. (b) One jumper suffices to fix the antenna violation if jumpers can be inserted at an arbitrary position of the wire segment.

In this paper, we consider the general case of inserting jumpers at arbitrary positions (e.g., in any position of a tree edge). We formulate the general jumper insertion for antenna avoidance (applicable at the routing stage) and/or fixing (applicable at the post-layout stage) as a tree-cutting problem. We show that the tree-cutting problem exhibits the properties of optimal substructures and greedy choices. With these properties, a greedy algorithm suffices to find an optimal solution [2]. Based on the theory, we present an $O(V \lg V)$-time exact jumper insertion algorithm that uses the minimum number of jumpers to fix the antenna violations in a routing tree with V vertices. Compared with the previous work in [3], our algorithm outperforms the method by large margins.

2. PROBLEM DEFINITION

To avoid/fix the antenna violation, we require that the total effective conductor connecting to a gate is less than or equal to a threshold, L_{max}. The threshold can be wire length limit, wire area limit, or any model of the strength of antenna effect caused by conductors. Typically, a net is modeled as a routing tree, where a node in the tree denotes a circuit terminal (a gate or a diffusion) and an edge denotes the interconnection between two circuit terminals. Since the interconnection connecting to a diffusion terminal will not cause any antenna violation, as explained in Section 1, we shall focus on those connecting to gate terminals.

Let $T = (V, E)$ be a routing tree, which can be a Steiner tree or a spanning tree of any form. The set V of nodes represents all gate terminals, the set E of edges denotes the wires connecting the circuit terminals, and an edge weight gives the measure of the wires with the same unit as L_{max}. For example, if L_{max} is a wire length limit, an edge weight denotes the wire length between two circuit terminals. If L_{max} is a wire area limit, the edge weight denotes the wire area. A gate will violate the antenna rule if the effective conductor incident on the gate (i.e., the *effective weight*—the sum of the weights of the edges incident on the corresponding node) is larger than L_{max}. To reduce the antenna effects on a gate, we can apply the technique illustrated in Figure 2 by adding a jumper on a wire connecting to the gate to reduce the effective conductor. This operation is modeled as adding a *cutting node* on the tree edge corresponding to the wire to reduce the effective edge weight associated with the gate node. As aforementioned, jumpers are implemented by vias which will consume silicon areas and reduce circuit performance. Therefore, it is desired to fix antenna violations by using the least jumpers. In other words, given a routing tree $T = (V, E)$ and an upper bound on the antenna L_{max}, we intend to add the minimum number of cutting nodes so that the effective edge weight associated with each node is smaller than L_{max}. Let $L(u)$ denote the sum of edge weights (lengths, wire areas, etc) between the node u and all its neighbors. We formulate the problem of jumper insertion on a routing tree for antenna avoidance/fixing as a tree cutting problem as follows:

- Problem *JITA* (*Jumper Insertion on a Routing Tree for Antenna Avoidance*): Given a routing tree $T = (V, E)$ and an upper bound L_{max}, find the minimum set C of cutting nodes, $c \neq u$ for any $c \in C$ and $u \in V$, so that $L(u) \leq L_{max}$, $\forall u \in V$.

Note that the routing tree in this formulation can be a Steiner tree or a spanning tree which represents a net in any layout design stage, e.g., a net to be globally routed, a net after detailed routing (in the post-layout stage). Therefore, the JITA problem is applicable to the antenna estimation in the global/detailed routing stage and the antenna violation fixing in the post-layout stage.

3. ALGORITHM FOR FINDING THE MINIMUM $|C|$

For the JITA problem, we present in this section an $O(V \lg V)$-time exact algorithm, named *BUJI* (*Bottom Up Jumper Insertion*), for finding the minimum cutting set C for a given routing tree $T = (V, E)$ with V nodes. (Note that we use V to denote the *set* or the *number* of nodes in a routing tree, which is common in the community of algorithms; its meaning is clear from the context.) Algorithm BUJI is summarized in Figure 4. To simplify the presentation, we assume that the antenna bound L_{max} is measured by wire length. Let $l(e)$ (or $l(u, v)$) be the length (i.e., weight) of the edge $e = (u, v)$ in T. In the BUJI algorithm, we add the cutting nodes into the original tree in a bottom-up manner. We first define a *subleaf* node as follows:

Definition 1. A *subleaf* is a node for which all its children are leaf nodes, and all the edges between it and its children have lengths $\leq L_{max}$.

We derive the algorithm based on the following two steps:

- Step 1 (lines 2–12 of Algorithm BUJI): We deal with every leaf node.

  In this case, our main goal is to prevent every leaf node from antenna violation. Obviously, if we have dealt with a leaf node, we need not consider it any more. Therefore, line 3 of the BUJI algorithm marks these nodes to make sure that every leaf node is processed only once. If $l(u, p(u)) \leq L_{max}$, the leaf node u satisfies the antenna rule. Thus, we need not insert any cutting nodes. However, if $l(u, p(u)) > L_{max}$, we must insert at least one cutting node to satisfy that $L(u) \leq L_{max}$. For this case, we can further divide it into two subcases as follows:

  1. $u \in C$:

     In this subcase, we need at least one cutting node to satisfy the rule that $L(u) \leq L_{max}$. We claim that $l(u, c) = l(u, p(u))$ (and thus $l(c, p(u)) = 0$) gives the best position for inserting the cutting node (the proof is given in the next section); see Figure 5(a) for an illustration. Therefore, we add c into C and cut the edge $e(u, p(u))$ from the original tree T (lines 5–8).

  2. $u \notin C$:

     In this subcase, we need at least one cutting node to prevent u from antenna violation. We claim that $l(u, c) = L_{max}$ (and thus $l(c, p(u)) = l(u, p(u)) - L_{max}$) gives the best position for inserting the cutting node (the proof is given in the next section); see Figure 5(b) for an illustration. Therefore, we add c into C, add c into V, and cut the node u and edge $e(u, c)$ from the original tree T (lines 9–12).

- Step 2 (lines 13–19 of Algorithm BUJI): We deal with every subleaf node.

  In this case, our main goal is to prevent every subleaf node from antenna violation. Moreover, we delete some nodes and edges to make each subleaf node as a leaf node. We classify the subleaf nodes into two categories by the sum of lengths between the node and its children. Let u_p be a subleaf node and $u_i, \forall 1 \leq i \leq k$, be its children. Let $totallen = \sum_{i=1}^{k} l(u_i, u_p)$.

  1. Case 1: $totallen \leq L_{max}$

     We use Subroutine LessEqual to deal with this case. If u_p and its children form an isolated component, they must satisfy the antenna rule, and thus we are done with the subroutine. If $totallen + l(u_p, p(u_p)) \leq L_{max}$, u_p will not violate the antenna rule. Therefore, we simply cut u_p's children from the original tree to make u_p as a leaf node

(lines 3–5 in LessEqual). Otherwise, we must add at least one cutting node c to prevent u_p from antenna violation. We claim that $l(c, u_p) + totallen = L_{max}$ gives the best position for inserting the cutting node; see Figure 6(a). Therefore, we add c into C, and cut u_p and all its children from the original tree T (lines 6–9 in LessEqual).

2. Case 2: $totallen > L_{max}$

   We use Subroutine More to deal with this case. We first sort edges from u_p and its children by their lengths. Let $A[i] = l(u_i, u_p), \forall 1 \le i \le k$. Then we find the maximum number s such that $\sum_{j=1}^{s} A[j] \le L_{max}$. We claim that $c_{s+1}, \ldots, c_k$ on edge $e(u_i, u_p)$ with $l(c_i, u_p) = 0$ (and thus $l(c_i, u_i) = l(u_p, u_i)$) give the best positions for inserting the cutting nodes; see Figure 6(b). Therefore, we add $c_{s+1}, \ldots, c_k$ into C, and cut $u_{s+1}, \ldots, u_k$ from the original tree T (lines 1–9 in More). Moreover, we call subroutine LessEqual to further reduce u_p into a leaf node (line 10 in More).

When the total length of the tree T is $\le L_{max}$, Algorithm BUJI terminates and C is a cutting set of the minimum size.

Algorithm: $BUJI(T, L_{max}, t_{total}, C)$
Input: $T = (V, E)$ /* The given tree. */
L_{max} /* Upper Bound on antenna */
C /* Cutting set */
t_{total} /* Total Edge Length in T */

```
1   while (t_total > L_max)
2     for each leaf node u ∈ T not having been processed
3       Mark u as processed;
4       if l(u, p(u)) > L_max
5         if u ∈ C
            Let c be the node between u and p(u) with
            l(u, c) = l(u, p(u)) and l(c, p(u)) = 0;
6           C ← C ∪ {c};
7           t_total ← t_total − l(u, p(u));
8           T(V, E) ← T(V − {u}, E − {e(u, p(u))});
9         else
            Let c be the node between u and p(u) with
            l(c, u) = L_max and l(c, p(u)) = l(u, p(u)) − L_max;
10          C ← C ∪ {c};
11          t_total ← t_total − L_max;
12          T(V, E) ← T(V + {c} − {u}, E − {e(u, c)});
13    for each subleaf node u_p ∈ T
14      Let u_1, u_2, ..., u_k denote all children nodes of u_p.
15      totallen ← Σ_{i=1}^{k} l(u_p, u_i);
16      if totallen ≤ L_max
17        LessEqual(T, t_total, C, u_p, totallen, L_max) ;
18      else
19        More(T, t_total, C, u_p, totallen, L_max);
```

Figure 4: Algorithm BUJI deals with the leaf nodes first, and then call Subroutines Equal, Less, and More to deal with the subleaf nodes.

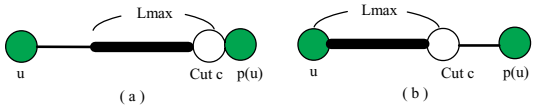

Figure 5: Explanation of lines 2–12 in the BUJI algorithm. (a) Case $u \in C$. (b) Case $u \notin C$.

4. PROOF OF THE OPTIMALITY OF $|C|$

Algorithm BUJI is greedy in nature. To prove that Algorithm BUJI finds the optimal cutting set (of the minimum size), therefore, we can show that the JITA problem exhibits *optimal substructure* and has the *greedy-choice property* [2]. A problem exhibits *optimal substructure* if an optimal solution to the problem contains within it optimal solutions to the subproblems; a problem has the *greedy-choice property* if a globally optimal solution can be arrived at by making a locally optimal (greedy) choice [2]. Due to the limitation of space, we shall omit several proofs for lemmas.

THEOREM 1. *The JITA problem exhibits optimal substructure.*

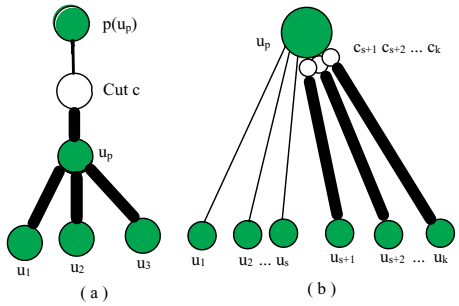

Figure 6: (a) Illustration of the *LessEqual* Subroutine. Here, $l(u_p, u_1) + l(u_p, u_2) + l(u_p, u_3) + l(u_p, c) = L_{max}$. (b) Illustration of the *More* Subroutine.

Subroutine: LessEqual(T, t_{total}, C, u_p, $totallen$, L_{max})

```
1   if p(u_p) does not exist
2     return
3   if totallen + l(u_p, p(u_p)) ≤ L_max
4     t_total ← t_total − totallen;
5     T(V, E) ← T(V − ∪_{i=1}^{k} {u_i}, E − ∪_{i=1}^{k} {e(u_i, u_p)});
6   else
      Let c be the node on e(u_p, p(u_p))
      with l(c, u_p) + totallen = L_max;
7     C ← C ∪ {c} ;
8     t_total ← t_total − L_max;
9     T(V, E) ← T(V − ∪_{i=1}^{k} {u_i} + {c} − {u_p},
                  E − ∪_{i=1}^{k} {e(u_i, u_p)} − {e(u, c)});
```

Figure 7: Compute the case when $totallen \le L_{max}$.

Now we show that the JITA problem has the greedy-choice property, and Algorithm BUJI finds the best solution in each step. First, we show that Algorithm BUJI has greedy choice property among all leaf nodes. Then, we show that BUJI has greedy choice property among all subleaf nodes.

LEMMA 1. *Lines 2–12 of Algorithm BUJI finds the best cutting set so that every leaf node u satisfies the antenna rule (i.e., $L(u) \le L_{max}, \forall$ leaf nodes u).*

We proceed to show that lines 13–19 in BUJI finds the best cutting set for each subleaf node u_p. In this step, we classify the subleaf nodes into two categories based on the sum of lengths between u_p and its children u_i: $\sum_{i=1}^{k} l(u_p, u_i) \le L_{max}$, and $\sum_{i=1}^{k} l(u_p, u_i) > L_{max}$. Therefore, we show that each case is with the greedy-choice property, and we find the best cutting set in each case.

Subroutine: More(T, t_{total}, C, u_p, $totallen$, L_{max})

```
1   Let A[i] ← l(u_i, u_p), ∀1 ≤ i ≤ k.
2   Sort A in non-decreasing order;
3   s ← 1;
4   while (Σ_{j=1}^{s+1} A[j] ≤ L_max)
5     s ← s + 1;
    Let c_{s+1}, ..., c_k be the nodes between u_p and u_{s+1}, ..., u_k
    with l(c_i, u_p) = 0 and l(c_i, u_i) = l(u_p, u_i), ∀ s + 1 ≤ i ≤ k;
6   C ← C ∪ {c_i} ∀ s + 1 ≤ i ≤ k ;
7   T(V, E) ← T(V − ∪_{i=s+1}^{k} {u_i}, E − ∪_{i=s+1}^{k} {e(u_i, u_p)});
8   minuslen ← Σ_{i=s+1}^{k} A[i];
9   t_total ← t_total − minuslen;
10  LessEqual(T, t_total, C, u_p, totallen − minuslen);
```

Figure 8: Compute the situation that $totallen > L_{max}$.

LEMMA 2. *Subroutine LessEqual finds the best cutting set so that every subleaf node u_p satisfies the antenna rule (i.e., $L(u_p) \leq L_{max}, \forall$ subleaf nodes u_p satisfying $\sum_{i=1}^{k} l(u_p, u_i) \leq L_{max}$, where $u_p = p(u_i)$).*

LEMMA 3. *Subroutine More finds the best cutting set so that every subleaf node u_p satisfies the antenna rule (i.e., $L(u_p) \leq L_{max}, \forall$ subleaf nodes u_p satisfying $\sum_{i=1}^{k} l(u_p, u_i) > L_{max}$, where $u_p = p(u_i)$).*

Based on the above theorem and lemmas, we have the following theorem:

THEOREM 2. *The BUJI algorithm finds an optimal solution.*

5. COMPLEXITY ANALYSIS

We analyze the time and space complexity of Algorithm BUJI in this section.

5.1 Time Complexity

In the BUJI Algorithm, we use the bottom-up method to find the optimal solution for the routing tree. We consider each leaf and each subleaf only once. Since every node in the tree might be a subleaf and might be cut into a leaf, we traverse each node at most twice. When we traverse a leaf node, we need only constant time. When we traverse subleaf nodes using Subroutine LessEqual, we also need constant time only. But in Subroutine More, we need to sort the lengths between subleaf and its children. The only condition that a node would be sorted is that it is a child of a subleaf node. Moreover, every node in the tree can be a subleaf node only once. Therefore, every node in the tree can be sorted at most once. In the $O(V \lg V)$-time sorting process, we can interpret that each node is computed $O(\lg V)$ times on average. Thus, when traversing each node, it would be computed at most $O(\lg V)$ times. To sum up, we traverse each node at most twice, and in each traversal, we compute each node at most $O(\lg V)$ times. Therefore, the total time complexity of the BUJI algorithm is $O(V \lg V)$.

5.2 Space Complexity

All we need to save are the tree T and the cutting set C. A tree needs only $O(V + E)$ space. Moreover, according to the algorithm, we add at most two cutting nodes for each edge. Therefore, we need $O(E)$ space to keep the set C. Since $O(E) = O(V)$ in a tree, the total space complexity is $O(V)$.

THEOREM 3. *Algorithm BUJI optimally solves the JITA problem in $O(V \lg V)$ time using $O(V)$ space.*

6. EXPERIMENTAL RESULTS

We implemented the BUJI algorithms in the C++ language on a 2.4 GHz Intel Pentium PC with 256 MB memory under the Windows XP operating system.

For the JITA problem, we compared our algorithm with the ISPD-04 work [3].

For comparative study, we first randomly generated tree nodes on grid planes of the dimension $10^5 \mu m \times 10^5 \mu m$, assuming that each node is a gate terminal. Then, we constructed a routing tree (a minimum spanning tree) for the nodes. We performed the following two experiments for our BUJI algorithm,and the ISPD-04 work [3]:

(1) First, for a given routing tree, we find the minimum number of jumpers required for fixing all antenna violations for various L_{max} values.

(2) Second, giving L_{max} as a constant, we find the running times for the algorithms and the heuristics to fix all antenna violations for routing trees with various numbers of nodes.

Table1 shows the number of jumpers required for fixing all antenna violations for a routing tree with 500000 nodes by changing L_{max} from 100 μm to 800 μm. Note that the L_{max} range is typical for 90 nm to 250 nm CMOS technologies. Column 1 gives the L_{max} value, and Columns 2 and 3 list the numbers of jumpers required ($\#J$) for fixing the antenna violations for each L_{max} for the BUJI algorithm and the ISPD-04 work [3], respectively. Columns 4 gives the percentage of additional jumpers required ($\%More$) for the ISPD-04 method over the BUJI algorithm to fix all antenna violations, i.e., $\%More$ = (#Jumpers of the heuristic − #Jumpers of BUJI)/ #Jumpers of BUJI. From the table, we have the following finding:

- It is not surprised that BUJI performs much better than the ISPD-04 method [3]. The phenomenon can be explained as follows: When we deal with nodes in a bottom-up manner, BUJI always pushes the jumper upward until at a position that just satisfies the antenna rule, adding more freedom and thus reducing the chance of antenna violations for the upper nodes. Therefore, BUJI can save a significant number of jumpers for antenna avoidance/fixing. In contrast, the ISPD-04 method [3] does not have such an optimization scheme.

L_{max} (um)	BUJI	ISPD04	
	$\#J$	$\#J$	$\%More$
100	517851	729048	+40.8%
200	232063	336084	+44.8%
300	98465	129891	+31.9%
400	35178	41136	+16.9%
500	9873	10658	+7.95%
600	1925	1978	+2.75%
700	319	321	+0.63%
800	40	40	0.00%

Table 1: Comparisons of the numbers of jumpers required for BUJI and ISPD-04 for fixing all antenna violations based on a routing tree of 500000 nodes.

Table 2 shows the CPU times required for antenna fixing on routing trees of the number of nodes ranging from 100000 to 900000, based on $L_{max} = 50\mu m$. Column 1 gives the numbers of nodes in the routing trees. Because the numbers of nodes are so huge, the program spent most of the CPU times on reading the input files. Therefore, we list the CPU times for reading the input files and running the algorithm/heuristic separately in order to examine the time complexity more closely. The 2nd column ($File$) gives the CPU times for reading the input files. The ($Main$) columns in each algorithm/heuristic give the respective CPU times for executing the main body of the algorithm/heuristic. As shown in the table, the empirical running times for the two methods are close to linear. (Note that the time complexities of the two methods are all $O(V \lg V)$.) In particular, BUJI requires only 3.8 sec to find an optimal solution for a routing tree of 0.9 million nodes. Therefore, the BUJI algorithm can handle a test case of a very huge number of nodes in very short time.

node number		BUJI	ISPD-04
	File (s)	Main (s)	Main (s)
100000	1.062	0.295	0.284
200000	2.103	0.684	0.641
300000	3.208	1.041	1.031
400000	4.228	1.451	1.451
500000	5.363	1.893	1.872
600000	6.425	2.345	2.356
700000	7.498	2.808	2.798
800000	8.591	3.292	3.292
900000	9.654	3.828	3.754

Table 2: Comparisons of the CPU times required for BUJI and ISPD-04 to fix the antenna violations, based on $L_{max} = 50\mu m$.

7. CONCLUSION

We have presented a loglinear-time exact jumper insertion algorithms for avoiding/fixing antenna violations on routing trees. Empirical results have shown that our algorithm approach linear and obtain solutions of very high quality. Our work can apply to any routing trees (could be a net to be globally routed or a net after detailed routing) and thus readily be incorporated into a global router for antenna effect avoidance or a post-layout optimizer for antenna violation fixing.

8. REFERENCES

[1] P. H. Chen, S. Malkani, C.-M. Peng, and J. Lin, "Fixing antenna problem by dynamic diode dropping and jumper insertion", *Proc. ISQED*, pp 275-282, 2000.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein "Introduction to Algorithms" *McGraw-Hill Book Company*, 2nd Edition, 2001.

[3] T.-Y. Ho, Y.-W. Chang, and S.-J. Chen, "Multilevel routing with antenna avoidance," *Proc. ISPD*, April 2004.

[4] S. Krishnan, et. al., "Assessment of charge-induced damage to ultra-thin gate MOSFETs", *Proc. ITEM*, pp. 445-448, 1997.

[5] H. Shin, C. -C. King, and C. Hu, "Thin Oxide Damage by plasma etching and ashing process", *Proc. IRPS*, 1992.

[6] H. Watanabe, et.al., "A wafer level monitoring method for plasma-charging damage using antenna PMOSFET test structure," *IEEE Trans. Semiconductor manufacturing*, vol. 10, no. 2, May. 1997.