Optimal Jumper Insertion for Antenna Avoidance Considering Antenna Charge Sharing

Jia Wang and Hai Zhou, *Senior Member, IEEE*

***Abstract*—Antenna effect may damage gate oxides during a plasma-based fabrication process. The antenna ratio of total exposed antenna area to total gate oxide area is directly related to the amount of damage. Jumper insertion is a common technique applied at routing and post-layout stages to avoid and to fix the problems caused by the antenna effect. This paper presents an optimal algorithm for jumper insertion under the ratio upper bound. It handles Steiner trees with obstacles. The algorithm is based on dynamic programming while working on free trees. The time complexity is $O(\alpha|V|^2)$ and the space complexity is $O(|V|^2)$, where $|V|$ is the number of nodes in the routing tree and α is a factor depending on how to find a nonblocked position on a wire for a jumper.**

***Index Terms*—Antenna effect, jumper insertion.**

I. Introduction

ANTENNA EFFECT is a phenomenon in very large scale integration (VLSI) fabrication where a current caused by a plasma process flows through the gate oxides and damages them. It reduces both manufacturing yield and product reliability, which are among the most important issues with the rapid scaling down of the VLSI feature sizes. The relationship between the amount of damage and the antenna ratio has been studied for a long time [3]–[5]. Generally speaking, a wire segment acting as an antenna may collect a charging current when exposed to plasma. If the segment only connects to the gate oxides but not diffusions, a voltage potential may build up and a discharging current tunneling through the gate oxides will form under the potential and damage the gate oxides. The damage can be observed via the drift of threshold voltage; the spatial variations of plasma stress across the wafer may cause variations in the threshold voltages with spatial correlations. Besides the processing parameters, which are constant, the antenna ratio of the total exposed antenna area to the total gate oxide area determines the amount of voltage potential and, thus, the damage: A smaller antenna ratio results in less damage. Antenna rules are commonly enforced as upper bounds on the antenna ratio in the design rules [6].

Correcting the antenna problem after the placement and routing stage is feasible and effective [7]–[9]. If a wire segment violates the antenna rule, either jumpers are inserted to break the wire segment or a protection diode is added to the unused area of the chip and connected to the wire segment to form a low impedance discharging path. However, as indicated by the works in [1] and [2], a correction after the placement and routing may not be effective or even possible beyond the 0.13-μm technology. The main reason is that with the scaling down of the VLSI feature sizes, the antenna rules become more stringent. The number of wire segments violating the rules and, thus, the numbers of the jumpers and the protection diodes increase dramatically, which is not practical, considering the limitation of the unused chip area and the routing resource. Therefore, considering the antenna effect in an earlier stage and planning ahead are a must to avoid the problems.

Two recent works [10], [11] considered the antenna effect during the routing stage by combining the jumper insertion to a layer assignment. In these approaches, the jumper insertion guides the layer assignment algorithms by predicting the jumper positions. Assigning a wire segment to a layer has the same effect of inserting a jumper if a discharging path to the source is formed. If for some reason, e.g., limited routing resource, such layer assignment is not feasible, a jumper must be inserted. Two more works [12], [13] focused on the jumper insertion problem itself. The latter one provided an exact algorithm to solve it in general routing trees with obstacles. In the works in [10], [12], and [13], a random discharging model was used and the upper bound on the total exposed antenna area connected to the gate oxides was controlled. In [11], although the same bound was controlled, Wu *et al.* pointed out that when multiple sinks are presented, the algorithm will be conservative and not optimal. More specifically, since there may be multiple gate oxides connected to one wire segment, the discharging current is "shared" among those gates. Using a small upper bound on the total exposed antenna area would overconstrain the routing, while using a large upper bound on the total exposed antenna area would result in a huge number of wire segments violating the antenna ratio rule. In addition, when gates are commonly sized for performance and power consumption during design, determining a proper upper bound on the total exposed antenna area could be more difficult. Therefore, the upper bound on antenna ratio, which is the most important rule for the antenna effect, should be directly addressed to improve the accuracy in both antenna planning in routing and antenna fixing in post-layout stages.

In this paper, we present an optimal algorithm to solve the jumper insertion problem under the upper bound of the antenna ratio. Our algorithm handles general routing trees, i.e., Steiner trees, as well as obstacles. Since we directly consider the ratio upper bound, we get a better result for antenna avoidance. We

Manuscript received July 5, 2006; revised October 2, 2006. This work was supported by the National Science Foundation under Grant CCR-0238484. This work was presented in part at the Design Automation Conference, 2006. This paper was recommended by Associate Editor J. Hu.

The authors are with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208 USA (e-mail: haizhou@eecs.northwestern.edu).

Digital Object Identifier 10.1109/TCAD.2007.893551

first formulate the RatioJI problem that models the jumper insertion under ratio upper bound. Then, we solve it via dynamic programming by the RatioPart algorithm. The time complexity is $O(\alpha|V|^2)$ and the space complexity is $O(|V|^2)$, where $|V|$ is the number of the nodes in the routing tree and α is a factor depending on how to find a nonblocked position on a wire for a jumper. In our experiments with and without obstacles, we have an α equal to 1. Moreover, our dynamic programming algorithm works on free trees. It is different from the classical dynamic programming approaches in the VLSI CAD area [17], [18], which work on rooted trees. We believe that the general framework is valuable for other problems on free trees and that it is possible to exploit this characteristic to reduce the practical running time via heuristics that determine the order for performing operations dynamically.

The rest of this paper is organized as follows. In Section II, backgrounds on antenna effect are introduced. The RatioJI problem is formulated in Section III. An algorithm to solve the problem is presented in Section IV. After the experimental results given in Section V, Section VI concludes this paper.

II. Antenna Effect

A detailed overview of the antenna effect can be found in the work of Shin and Hu [3]. We briefly introduce the backgrounds here.

VLSI chips are commonly fabricated layer by layer. Since interconnect networks consist of wires and vias on and between different layers, wire segments are formed during the fabrication. Some of them connect to gate inputs, which are gate oxides, and others connect to gate outputs, which are diffusions. During radio frequency (RF) plasma processes, the exposed wire segments act as antennas. They collect ion and electron currents from the plasma. Since a wire segment also acts as a capacitor, if the two currents do not cancel each other through every RF cycle and the wire segment does not connect to a diffusion, charging on the capacitor happens and an antenna voltage V_g will build up. When the wire segment only connects to the gate oxides, V_g reaches a steady state when the Fowler–Nordheim (FN) tunneling current through the gate oxides balances the plasma caused charging current. In such a steady state, V_g is also the gate voltage on all the gate oxides connected to the wire segment. The charging current density J_p is a function of the antenna voltage, and the tunneling current density J_{FN} is a function of the gate voltage when the technology parameters are given as constants. Since both the antenna voltage and the gate voltage are V_g, the following equation shows the relationship between the antenna ratio and the current densities:

$$\text{antenna ratio} \overset{\triangle}{=} \frac{\text{total exposed antenna area}}{\text{total gate oxide area}} = \frac{J_{\mathrm{FN}}(V_g)}{J_p(V_g)}.$$

As plotted in [3, Fig. 12], the charging current decreases with the increase of V_g, but the tunneling current, which damages the gate oxide, increases with the increasing of V_g. A higher antenna ratio means a larger V_g and, thus, a larger J_{FN} and more damage.

The methods to compute the exposed antenna area are different for different plasma-based manufacturing processes. According to the work of Shirota *et al.* [7], there are three types. The first is the conductor layer pattern etching process where the perimeters of the wires are exposed. Therefore, the exposed area is computed as the perimeter length of conductor layer patterns. The second is the ashing process where the area of the wires is exposed. Thus, the exposed area is the area of the conductor layer patterns. The third is the contact etching process where the area of the contacts on the lower conductor layer is exposed. Therefore, the exposed area is the total area of the contacts. Our problem formulation is general enough to handle all these types.

III. Problem Formulation

Considering the jumper insertion on a general routing tree for an upper bound R on the antenna ratio, let $T = (V, E)$ be the routing tree, where V is the set of nodes representing the gates as well as the Steiner points and E is the set of tree edges representing the wires connecting those nodes. A function g is defined on every node $v \in V$. If v represents a gate, $g(v) > 0$ gives the gate oxide area; if v does not represent a gate but a Steiner point, let $g(v) = 0$. A function l is defined on an edge segment s: $l(s) \geq 0$ is the exposed antenna area of s.

Jumpers will be inserted on edges to form cuts. For an edge $e = (u, v) \in E$, a cut c_e divides e into edge segments. The size of the cut c_e, written as $|c_e|$, is defined as the number of the jumpers inserted. Since there is no gate within the edge, it is obvious that at most two jumpers are enough to satisfy the antenna rule. Thus, c_e is written as

$$c_e = (p_0 = u, p_1, \ldots, p_{k+1} = v), \qquad k = 0, 1, \text{ or } 2$$

where $(p_{i-1}, p_i),\ i = 1, 2, \ldots, k+1$ are the edge segments and $|c_e| = k$. It is not always possible to insert a jumper at any position. Various reasons introduce obstacles along the wire. For example, the top layers may be occupied by other nets. To model the obstacles on a particular wire, which are the positions that a jumper cannot be inserted, we use $\mathcal{C}_e$ to denote the set of allowed cuts on an edge e.

By assigning one cut to each edge in the tree, T is partitioned into connected components. The partitioning is given by the set of the cuts: $C = \{c_e : e \in E\}$. A partitioning C is called feasible if $c_e \in \mathcal{C}_e$ for every $e \in E$. Obviously, the number of all the jumpers inserted is $\sum_{e \in E} |c_e|$. For a connected component S, let $g(S)$ be the total gate oxide area and $l(S)$ be the total exposed antenna area. A feasible partitioning is valid if for every connected component S, either $g(S) = 0$ or $l(S)/g(S) \leq R$. The reason is that, according to Section II, for all the gate oxides in a specific connected component, the tunneling current densities are the same since the gate voltages are all equal to the antenna voltage. Intuitively, the plasma charging current is "shared" among those gates. A connected component can survive larger total exposed antenna area if the total gate oxide area is larger. Note that it is possible to have no valid partitioning because of the obstacles.

Since inserting the jumpers will degrade the performance and manufacturing yield, it is preferred to find the minimal number of jumpers to meet a specific ratio upper bound. The corresponding partitionings are called the optimal partitionings. The optimal jumper insertion under the ratio upper bound is formulated as the following RatioJI problem.

Problem 1 (RatioJI): Suppose that $T = (V, E)$ is a tree, functions g and l model the gate oxide area and the exposed antenna area, respectively, and set $\mathcal{C}_e$ models the obstacles on a wire e where jumpers cannot be inserted. For a ratio upper bound R, find an optimal partitioning, which is a partitioning C with $c_e \in \mathcal{C}_e, \forall e \in E$ and a minimal number of jumpers such that for a connected component S, either the total gate oxide area $g(S)$ is 0 or the ratio of total exposed antenna area to total gate oxide area $l(S)/g(S) \leq R$.

The RatioJI problem models both antenna fixing and antenna planning under various antenna ratio models. For antenna fixing after the layer assignment, the problem can be solved at each layer to meet the antenna ratio rule on that layer with proper parameters. For antenna planning in routing, the layer assignment can be obtained in an algorithm similar to those in [10] and [11] by predicting the jumper positions via solving the RatioJI problem. Although the predictions may be inaccurate, the previous works [10], [11] showed that the number of jumpers required for antenna fixing later could be reduced.

Two commonly used models of antenna ratio are the partial antenna ratio (PAR) and cumulative antenna ratio (CAR) [11]. In PAR, the antenna ratio bound is applied to the antenna ratio computed at a specific layer for a specific wire segment at a time. Different layers may have different bounds. The RatioJI problem models the PAR by constructing T to represent the part of a routing tree, including that wire segment and all the interconnects and gates connected to it in lower layers which are already fabricated. Other parameters are chosen accordingly. In CAR, the antenna ratio is defined as the ratio of the cumulative antenna area to its connected gate area. The RatioJI problem models the CAR by constructing T to represent the cumulative antenna area and its connected gates and l to represent the antenna area contributing to the cumulative antenna area.

IV. Optimal Jumper Insertion

A. Algorithm Overview

We solve the RatioJI problem by dynamic programming. Classical dynamic programming algorithms usually work on a rooted tree [17], [18]. However, a routing tree T is a free tree without a root. Our algorithm will process the edges one by one in some order. We briefly describe our algorithm here, while the definitions of terminologies and the details of algorithm will be presented in later sections.

Our algorithm consists of two stages. In the first stage, dominant solutions are generated bottom-up. At any step, the edges are divided into two groups: processed or unprocessed. Dominant partial solutions are maintained for every tree in the forest formed by connecting all nodes via the processed edges. At the beginning, no edge is processed. Thus, every node is a tree by itself and has only one partial solution. The algorithm processes one edge at a time by combining two trees with an unprocessed edge and computing the dominant partial solutions of the new tree based on those of the two old trees. An edge can be processed only if it is the only unprocessed edge incident to one tree. An example is shown in Fig. 1. Starting from the process in Fig. 1(a), there are three possible edges to be processed next, as shown in Fig. 1(b)–(d). Note that the edge (e, h) cannot be processed at the current step. When every edge is processed at the end, the forest contains one tree, which is T itself, and all the dominant solutions are generated.

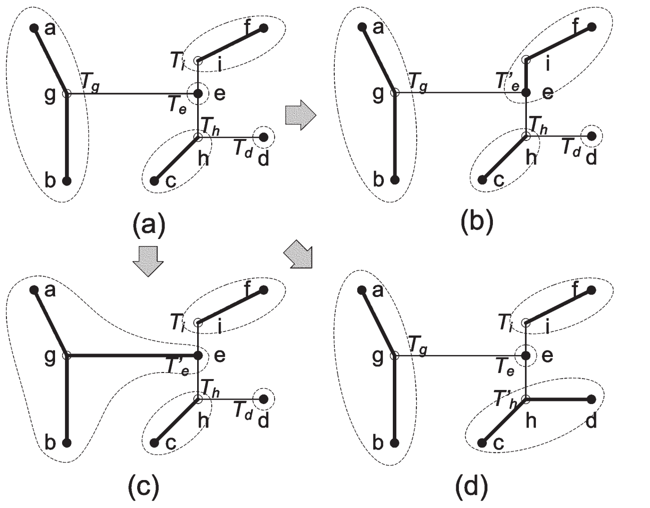


Fig. 1. Process one edge by combining two trees. Bold edges are processed; others are unprocessed. Solid nodes are gates; others are Steiner points. (a) Edges (a, g), (b, g), (f, i), and (c, h) are processed while the dominant partial solutions are maintained on trees T_g, T_d, T_h, and T_i. One of the three edges, (e, i), (e, g), and (d, h), could be processed next. (b) (e, i) is processed by combining T_e and T_i into T'_e. (c) (e, g) is processed by combining T_e and T_g into T'_e. (d) (d, h) is processed by combining T_h and T_d into T'_h.

In the second stage, since there is one optimal partitioning among the dominant solutions if there exists a valid partitioning, we can either find an optimal partitioning or report that there is no valid partitioning. A top–down backtracking will construct the optimal partitioning if there is one.

B. Dominant Solutions

Denote the forest formed by connecting all nodes via the processed edges by F. Suppose there are m trees in F, written as $T_1, T_2, \ldots, T_m$. There are $m - 1$ unprocessed edges currently. When $m > 1$, every T_i has at least one unprocessed edge that connects to it because T is connected. Since an edge can be processed only if it is the only unprocessed edge to one tree, each tree T_i can only have one node u_i incident on the unprocessed edges.

For a tree T_i, partial solutions are the partitionings of T_i with the following properties. For every component S not containing u_i, the ratio upper bound must be satisfied, i.e., either $g(S) = 0$ or $l(S)/g(S) \leq R$. Denote the component containing u_i by S_i. If there is at least one unprocessed edge incident on u_i, the upper bound can be violated in S_i, which may be corrected when future combinations are executed. For a partition of T_i, let n be the number of the jumpers inserted. Define $x = l(S_i) - g(S_i)R$. Let I be 0 if $g(S_i) = 0$ and 1 if $g(S_i) > 0$.

The partial solution is represented by the triple (n, x, I). To ease the representation, we refer to the partial solution by the triple when there is no ambiguity.

A dominant relationship is defined among the partial solutions such that only necessary partial solutions are maintained for each tree T_i. For two different partial solutions $P_1 = (n_1, x_1, I_1)$ and $P_2 = (n_2, x_2, I_2)$ of one tree, we say that P_1 dominates P_2 if $(n_1 \leq n_2) \wedge (x_1 \leq x_2) \wedge (I_1 = I_2)$. The reason is that a partial solution with smaller n and x can always substitute another one of the same I in any optimal partitioning of T.

If there are no other partial solutions dominating (n, x, I), we call (n, x, I) a dominant partial solution. By denoting the number of nodes in T_i as N_i, the number of dominant partial solutions of T_i is upper-bounded by a linear function of N_i.

Lemma 1: There are at most $2N_i - 2$ jumpers on a tree T_i of N_i nodes, and thus, there are at most $4N_i - 2$ dominant partial solutions to maintain.

Proof: A tree T_i of N_i nodes has $N_i - 1$ edges. Since at most two jumpers are inserted per edge, there are at most $2N_i - 2$ totally.

For each $n \in \{0, 1, \ldots, 2N_i - 2\}$ and $I \in \{0, 1\}$, only one dominant partial solution (n, x, I) is maintained. Thus, there are at most $(2N_i - 1) \times 2 = 4N_i - 2$ dominant partial solutions to maintain. ■

When there is only one tree in the forest, i.e., $m = 1$, we call the dominant partial solutions of the tree dominant solutions. Note that the set of dominant solutions could be different with different orders to process the edges.

C. Generation of Dominant Solutions

In this section, we consider how to generate solutions for a new tree by assimilating one tree into another. Assume that there are more than one tree in the forest, i.e., $m > 1$. Consider an unprocessed edge $e = (u, v)$ connecting two trees T_u and T_v in the forest F. We process the edge e and combine T_u and T_v only when e is the sole unprocessed edge incident on u. Denote the new tree made up of T_u, T_v, and e by T'_v. The dominant partial solutions of tree T'_v are generated from those of the trees T_u and T_v, as stated in the following lemma.

Lemma 2: A dominant partial solution (n, x, I) of T'_v consists of a dominant partial solution (n_v, x_v, I_v) of T_v, a dominant partial solution (n_u, x_u, I_u) of T_u, and a cut c_e on e. The c_e and (n, x, I) must be one of the following cases.

1) If $c_e = (u, v)$, then $n = n_v + n_u$ and $x = x_v + x_u + l(u, v)$. The I is 0 if $I_v = I_u = 0$; otherwise, I is 1.
2) If $c_e = (u, p_1, v)$, then $n = n_v + n_u + 1$, $x = x_v + l(p_1, v)$, and $I = I_v$. The I_v and I_u must not be 1 at the same time. If $I_u = 1$, then c_e is the element in $\mathcal{C}_e$ satisfying $x_u + l(u, p_1) \leq 0$ with the minimal $l(p_1, v)$. If $I_u = 0$, then c_e is the element in $\mathcal{C}_e$ with the minimal $l(p_1, v)$.
3) If $c_e = (u, p_1, p_2, v)$, then $n = n_v + n_u + 2$, $x = x_v + l(p_2, v)$, and $I = I_v$. If $I_u = 1$, then c_e is the element in $\mathcal{C}_e$ satisfying $x_u + l(u, p_1) \leq 0$ with the minimal $l(p_2, v)$. If $I_u = 0$, then c_e is the element in $\mathcal{C}_e$ with the minimal $l(p_2, v)$.

```
Subroutine Combine
Inputs   $e = (u, v)$, $D_v$, and $D_u$.
Outputs  Updated $D_v$.
1  $D^* \leftarrow \emptyset$.
2  For each $(d_v, d_u)$ in the set $D_v \times D_u$:
3    $(n_v, x_v, I_v, B_v) \leftarrow d_v$; $(n_u, x_u, I_u, B_u) \leftarrow d_u$.
4    For each case in Lemma 2:
5      Compute $(n, x, I)$ and $c_e$.
6      $B \leftarrow B_v \cup \{(n_u, x_u, I_u, u, e, c_e)\}$.
7      Add $(n, x, I, B)$ to $D^*$.
8  Remove non-dominant partial solutions from $D^*$.
9  $D_v \leftarrow D^*$.
```

Fig. 2. Combine subroutine.

Proof: We proof the lemma by examining each case.

1) When $c_e = (u, v)$, it is straightforward that $n = n_v + n_u$, $x = x_v + x_u + l(u, v)$, and I is 0 if $I_v = I_u = 0$ or 1 otherwise.
2) When $c_e = (u, p_1, v)$, it is straightforward that $n = n_v + n_u + 1$, $x = x_v + l(p_1, v)$, and $I = I_v$. Denote the component containing u in the partial solution (n, x, I) by S. If $I_u = 1$, we have $g(S) > 0$. The following condition should be satisfied by p_1:
$$0 \geq l(S) - g(S)R = x_u + l(u, p_1).$$
We prove $I_v = 0$ by contradiction. If $I_v \neq 0$, we have $I_v = 1$ and
$$\begin{aligned} x &= x_v + l(p_1, v) \\ &\geq x_v + l(p_1, v) + (x_u + l(u, p_1)) \\ &= x_v + x_u + l(u, v). \end{aligned}$$
The (n, x, I) is dominated by the partial solution in case 1), which violates our assumption. Therefore, I_v should be 0 and c_e should be the element in $\mathcal{C}_e$ satisfying $x_u + l(u, p_1) \leq 0$ with the minimal $l(p_1, v)$.
If $I_u = 0$, we have $g(S) = 0$. Therefore, c_e should be the element in $\mathcal{C}_e$ with the minimal $l(p_1, v)$.
3) When $c_e = (u, p_1, p_2, v)$, it is straightforward that $n = n_v + n_u + 2$, $x = x_v + l(p_2, v)$, and $I = I_v$. Denote the component containing u in the dominant partial solution (n, x, I) by S. If $I_u = 1$, we have $g(S) > 0$. The following condition should be satisfied by p_1:
$$0 \geq l(S) - g(S)R = x_u + l(u, p_1).$$
Therefore, c_e should be the element in $\mathcal{C}_e$ satisfying $x_u + l(u, p_1) \leq 0$ with the minimal $l(p_2, v)$.
If $I_u = 0$, we have $g(S) = 0$. Therefore, c_e should be the element in $\mathcal{C}_e$ with the minimal $l(p_2, v)$. ■

Suppose D_v and D_u are the sets of the dominant partial solutions of T_v and T_u, respectively. We design the Combine subroutine, as shown in Fig. 2, based on Lemma 2. The Combine subroutine updates D_v to be the set of the dominant partial solutions of T'_v. A set B is stored for every dominant partial solution to enable the backtracking which is used to construct the optimal partitioning later.

```
ALGORITHM RatioPart
Inputs   $T$, $g$, $l$, $\mathcal{C}_e$'s, and $R$.
Outputs  Report an optimal partitioning if there is one.
1   Mark all the edges as unprocessed.
2   For each $u$ in $V$:
3     If $g(u) = 0$:
4       $D_u \leftarrow \{(0, 0, 0, \emptyset)\}$.
      Else:
5       $D_u \leftarrow \{(0, -g(u)R, 1, \emptyset)\}$.
6   While there is at least one unprocessed edge:
7     Pick an edge $e = (u, v)$ such that it is the
      only unprocessed edge on $u$.
8     Update $D_v$ by the Combine subroutine.
9     Mark $e$ as processed.
10  $w \leftarrow$ the last $v$ in the While loop.
11  $d_1 \leftarrow$ the element with the least $n$ in $D_w$
    whose $I = 0$.
12  $d_2 \leftarrow$ the element with the least $n$ in $D_w$
    whose $I = 1$ and $x \leq 0$.
13  If none of $d_1$ and $d_2$ exists:
14    Report there is no valid partitioning.
    Else:
15    $(n, x, I, B) \leftarrow$ the one with the smaller $n$.
16    ReportPart($B$).
```

Fig. 3. RatioPart algorithm.

In Fig. 2, D^* holds the candidates of the dominant partial solutions. Suppose T_u has N_u nodes and T_v has N_v nodes. The set D^* is implemented as an array of $4(N_u + N_v) - 2$ elements since there are at most $4(N_u + N_v) - 2$ combinations of n and I according to Lemma 1. Since D_u will not be modified by any following Combine calls, the position of element (n_u, x_u, I_u, B_u) in the array D_u is stored instead of the triple (n_u, x_u, I_u) on line 6. For the partial solutions generated on line 7, only the ones with different ns and Is are stored in D^*; if there are partial solutions with the same n and I, only the one with the least x is stored. Remaining nondominant partial solutions in D^* are removed on line 8. Then, line 7 takes $O(1)$ time, and both lines 1 and 8 take $O(N_u + N_v)$ time.

D. RatioPart Algorithm

Fig. 3 gives the RatioPart algorithm that solves the RatioJI problem. At the beginning, no edge is processed. The forest contains $|V|$ trees where every tree contains one node. There is one dominant partial solution on each tree since there is only one possible partitioning. According to this, the D_us are built on lines 2 to 5. The **While** loop on lines 6 to 9 processes the edges and updates the dominant partial solutions. The invariant of the loop is stated in the following lemma.

Lemma 3: At line 7 of the RatioPart algorithm, for every $1 \leq i \leq m$, all the unprocessed edges connecting to T_i are incident on one node u_i, and D_{u_i} has all the dominant partial solutions of T_i.

Proof: We prove the above invariant by induction.

Before the **While** loop on lines 6 to 9, every node v is a tree T_v by itself. All the unprocessed edges connecting to T_v are incident on v. The **For** loop on lines 2 to 5 initializes all the D_vs correctly.

Assume that the invariant holds on line 7. Denote the tree containing u by T_u and v by T_v, respectively. Denote the tree

```
Subroutine ReportPart
Inputs   The set $B$.
Outputs  Report the cuts corresponding to $B$.
1   For each $(n, x, I, u, e, c_e)$ in $B$:
2     Report the cut $c_e$ on edge $e$.
3     Find the element $(n_u, x_u, I_u, B_u)$ in $D_u$
      satisfying $n_u = n$, $x_u = x$, and $I_u = I$.
4     ReportPart($B_u$).
```

Fig. 4. ReportPart subroutine.

after processing e and combining T_u and T_v by T'_v. We only need to prove that the invariant holds for T'_v when the current loop finishes, since all the other trees and their dominant partial solutions remain unchanged.

Since all the unprocessed edges connecting to T_u are incident on u, the edge e is the only unprocessed edge connecting to T_u. After e is processed, there is no unprocessed edge connecting to T_u. Thus, all the unprocessed edges connecting to T'_v are incident on v.

We prove that the D^* in the Combine subroutine contains all the dominant partial solutions of T'_v by contradiction. Assume this is not the case. Suppose the dominant partial solution $(n', x', I') \notin D^*$, and it consists of some partial solution (n'_v, x'_v, I'_v) of T_v, some partial solution (n'_u, x'_u, I'_u) of T_u, and some cut c_e on e. Because D_v has all the dominant partial solutions of T_v, there must exist $(n_v, x_v, I_v) \in D_v$ such that $n_v \leq n'_v$, $x_v \leq x'_v$, and $I_v = I'_v$. Similarly, there is $(n_u, x_u, I_u) \in D_u$ such that $n_u \leq n'_u$, $x_u \leq x'_u$, and $I_u = I'_u$. Therefore, denoting the partial solution consisting of (n_v, x_v, I_v), (n_u, x_u, I_u), and c_e by (n, x, I), we have $n \leq n'$, $x \leq x'$, and $I = I'$. Since (n', x', I') is a dominant partial solution, it must be true that $n = n'$, $x = x'$, and $I = I'$. On the other hand, according to Lemma 2, either $(n, x, I) \in D^*$ or there exists some partial solution in D^* dominating (n, x, I). This contradicts that $(n', x', I') \notin D^*$ is a dominant partial solution.

Therefore, the invariant holds for T'_v after the current loop. ■

The code on lines 11 to 16 searches for an optimal partitioning in the set D_w. It is guaranteed to find one if there is one, which is stated in the following lemma.

Lemma 4: If there is a valid partitioning of T, then an optimal partitioning is presented in the dominant solutions D_w.

Proof: If there is a valid partitioning, there must be an optimal partitioning. Suppose the partial solution (n, x, I) of T_w is the optimal partitioning with the minimal x. We prove $(n, x, I) \in D_w$ by contradiction. According to Lemma 3, D_w has all the dominant partial solutions of T_w. If $(n, x, I) \notin D_w$, then there exists $(n', x', I') \in D_w$ dominating (n, x, I). Therefore, $n' \leq n$, $x' \leq x$, and $I' = I$. Thus, (n', x', I') is also valid. Since (n, x, I) is the optimal partitioning with the minimal x, we should have $n' \geq n$ and $x' \geq x$. Therefore, $n = n'$, $x = x'$, and $I = I'$, which contradicts that (n', x', I') dominating (n, x, I). ■

On line 15, (n, x, I) is the triple of the optimal partitioning, and the set B contains information to reconstruct the optimal partitioning. The subroutine ReportPart, as shown in Fig. 4, takes B as the parameter and recursively reports the optimal partitioning. The search on line 3 of the ReportPart subroutine takes a constant time since we implement D_u as an array, and the position of element with the triple (n_u, x_u, I_u) in the array is stored.

The correctness of the algorithm is given by the following theorem.

Theorem 1: The RatioPart algorithm terminates in finite time and gives an optimal partitioning if there is a valid one.

Proof: The **While** loop on lines 6 to 9 terminates because the number of unprocessed edges is limited and decreases by one each time. The ReportPart subroutine terminates because the number of edges is limited, and the cut on each edge is reported exactly once. It is straightforward that the other parts of the RatioPart algorithm terminate so the whole RatioPart algorithm terminates.

A valid partitioning is a partial solution (n, x, I) of T_w such that either $I = 0$ or $(I = 1) \wedge (x \leq 0)$. By searching all such partial solution in D_w, the RatioPart algorithm gives an optimal partitioning if there is a valid one according to Lemma 4. ■

In the current implementation, the order of the edges to be processed is determined statically in the RatioPart algorithm for simplicity. From an arbitrary node, a depth-first search (DFS) [19] on the routing tree is performed. Define the finishing time of edge (u, v) to be the finishing time of v where we assume that u is discovered prior to v. The edges are ordered according to their finishing times. For example, by a DFS of the tree in Fig. 1 starting from the node e, the nodes are discovered in the order e, g, a, b, h, c, d, i, and f and finished in the order a, b, g, c, d, h, f, i, and e. Thus, the order of the edges is (a, g), (b, g), (g, e), (c, h), (d, h), (h, e), (f, i), and (i, e). Note that it is possible to dynamically determine which edge to be processed next when there are multiple choices like the situation in Fig. 1. Although the set of the dominant solutions may be different for different choices, Theorem 1 ensures that an optimal partitioning will always be found if there is one. A good heuristic may benefit both the running time and the storage requirement. We leave this as a direction of future research.

E. Complexity of the RatioPart Algorithm

The space complexity of the RatioPart algorithm is stated in the following theorem.

Theorem 2: The space complexity of the RatioPart algorithm is $O(|V|^2)$.

Proof: In the RatioPart algorithm, D_u is stored for every u. Each D_u contains at most $4|V| - 2$ elements according to Lemma 1. Suppose there are m_u edges incident on u in T. Then, for every element in D_u, it contains a set B with at most m_u elements. Therefore, D_u requires at most $O(m_u|V|)$ storage, and the total storage required by all the D_us is

$$\sum_{u \in V} O(m_u|V|) = O\left(\left(\sum_{u \in V} m_u\right)|V|\right) = O\left(|V|^2\right).$$

For the Combine subroutine, D^* is an array of at most $O(|V|)$ elements, and every element requires at most $O(|V|)$ storage. Therefore, it requires at most $O(|V|^2)$ storage. For the ReportPart subroutine, since the depth of the recursion is at most $|V|$, it requires $O(|V|)$ storage. Therefore, the space complexity for the RatioPart algorithm is $O(|V|^2)$. ■

TABLE I
STATISTICS OF THE BENCHMARKS

name	# gates	# nodes	SMT	ratio
a100	100	154	79189	792
a1000	1000	1439	229604	230
a10000	10000	14542	721452	73
a20000	20000	28989	1019236	51
n2676	2676	3733	81180	31
n12052	12052	16447	243886	21
n22373	22373	30497	1264194	57
n34728	34728	47954	904740	27

To evaluate the time complexity, we assume that both functions g and l take a constant time to compute. The time complexity is stated in the following theorem.

Theorem 3: The time complexity of the RatioPart algorithm is $O(\alpha|V|^2)$, where α is a factor depending on how to find a nonblocked position on a wire for a jumper.

Proof: We use the potential method [19] to evaluate the total running time of the Combine subroutine when it is used in the RatioPart algorithm. In the Combine subroutine, suppose line 5 takes $O(\alpha)$ time, where α is a factor depending on how to find the element in $\mathcal{C}_e$ according to Lemma 2, i.e., how to find a nonblocked position on a wire for a jumper. Recall that N_v and N_u are the number of nodes of the trees T_v and T_u, respectively. Then, the loop from lines 2 to 7 takes $O(\alpha N_u N_v)$ time. As discussed in Section IV-C, lines 1 and 8 take $O(N_u + N_v)$ time. Therefore, the total running time of the Combine algorithm is $O(\alpha N_u N_v)$. Assume that the upper bound on the running time is $A\alpha N_u N_v$, where A is a constant.

For a forest F, define its potential to be

$$\Phi(F) = \frac{A}{2}\alpha \sum_{i=1}^{m} N_i^2$$

where N_i is the number of the nodes in the tree T_i. Suppose a forest F' is obtained after applying the Combine subroutine to process edge (u, v) and combine T_v and T_u. Denoting the running time for this Combine run by t, we have

$$\Phi(F') - \Phi(F) = A\alpha N_u N_v \geq t.$$

Since initially the potential is $(A/2)\alpha|V|$ and finally the potential is $(A/2)\alpha|V|^2$, the Combine subroutine takes at most

$$\frac{A}{2}\alpha|V|^2 - \frac{A}{2}\alpha|V| = O\left(\alpha|V|^2\right)$$

time when it is used in the RatioPart algorithm.

In the ReportPart subroutine, every D_u is searched, at most, once in the recursion. Therefore, line 16 in the RatioPart algorithm takes $O(|V|)$ time.

All the other parts in the RatioPart algorithm take at most $O(|V|^2)$ time. Therefore, the time complexity is $O(\alpha|V|^2)$. ■

TABLE II
RESULTS OF JUMPER INSERTION WITHOUT OBSTACLES

a100			a1000			a10000			a20000		
ratio	# cuts	time(s)	ratio	# cuts	time(s)	ratio	# cuts	time(s)	ratio	# cuts	time(s)
100	141	0.004	50	1406	0.078	10	14902	8.48	10	28807	39.8
200	138	0.004	150	607	0.037	30	10912	4.93	20	22873	26.9
400	92	0.002	200	197	0.018	50	5374	1.74	30	15020	12.8
600	41	0.002	210	121	0.016	60	2675	0.90	40	7091	4.78
700	18	0.002	220	54	0.016	70	376	0.51	50	444	1.97
800	0	<1ms	230	0	0.016	80	0	0.38	60	0	1.13

n2676			n12052			n22373			n34728		
ratio	# cuts	time(s)	ratio	# cuts	time(s)	ratio	# cuts	time(s)	ratio	# cuts	time(s)
10	3363	0.422	4	18048	16.1	10	36693	70.6	10	41853	88.7
15	2394	0.258	8	14403	9.83	30	18770	14.0	15	25652	37.7
20	1383	0.150	12	8593	4.48	40	10020	5.65	20	10804	13.4
25	508	0.074	16	3352	1.29	50	2833	2.74	23	3797	7.55
30	6	0.051	20	23	0.80	55	359	2.09	26	6	4.91
35	0	0.047	22	0	0.57	60	0	1.47	30	0	2.71

V. EXPERIMENTS

The experiments are conducted on a Linux workstation with dual 933-MHz Pentium III processors and 512-MB memory where the RatioPart algorithm is implemented in C++ and compiled with GCC 3.4. Since there is no previous work considering the upper bound of the antenna ratio with multiple sinks, we do not compare our results to others.

Two sets of benchmarks are used in the experiments. The benchmarks in the first set are four randomly generated ones containing 100, 1000, 10 000, and 20 000 gates, respectively, where the gates are randomly located in a 10 000 by 10 000 grid. The benchmarks in the second set are the four largest industrial benchmarks in the work [16]. For each benchmark, a Steiner tree is constructed as the routing tree using the algorithm in [15]. All the gate sizes are assumed to be one, and the exposed antenna area of a wire is computed as the wire length, which is the Manhattan distance between the two nodes it connects. The statistics of the benchmarks are shown in Table I. The column "name" shows the name of each benchmark where the benchmarks with names starting with "a" are the random generated ones and the benchmarks with names starting with "n" are the industrial ones. The column "# gates" shows the number of the gates in each benchmark. The column "# nodes" shows the number of the nodes on the routing tree, including the gates and the Steiner points. The column "SMT" shows the total wire length of the Steiner tree. The column "ratio" shows the antenna ratio of the routing tree when all the wires are exposed. For any antenna ratio upper bound larger than the number in this column, no jumper is needed.

We first run the RatioPart algorithm without obstacles. For each benchmark, six upper bounds of the antenna ratio are chosen representatively according to the "ratio" column in Table I. The results are reported in Table II. The "ratio" columns show the upper bounds used. The "# cuts" columns show the number of the jumpers inserted. The "time(s)" columns show the running time in seconds. It can be seen that the practical running times depend not only on the number of the nodes but also on the bound since different bounds will result in different numbers of partial dominant solutions to maintain on each tree. In Fig. 5, the trend of the decrease of the running time with the increase of the antenna ratio is shown by plotting the number of jumpers inserted and the running time in seconds versus different bounds for the benchmark a10 000. On the other hand, the quadratic theoretical time complexity can be verified in Fig. 6. There is one sample for each benchmark in the figure. Each sample is the one with the smallest bound among the six bounds for one benchmark. The slope of the best fitting line in the $\log - \log$ plot shows a practical running time of $\Theta(|V|^{1.85})$.

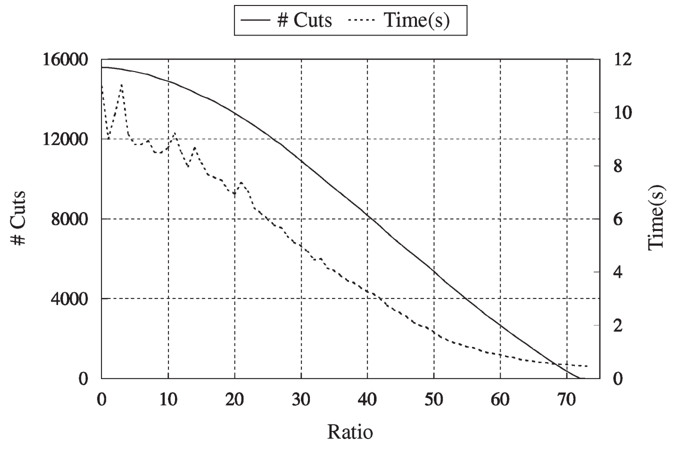


Fig. 5. Number of the cuts and the running time versus the antenna ratio bound for a10 000 without obstacles.

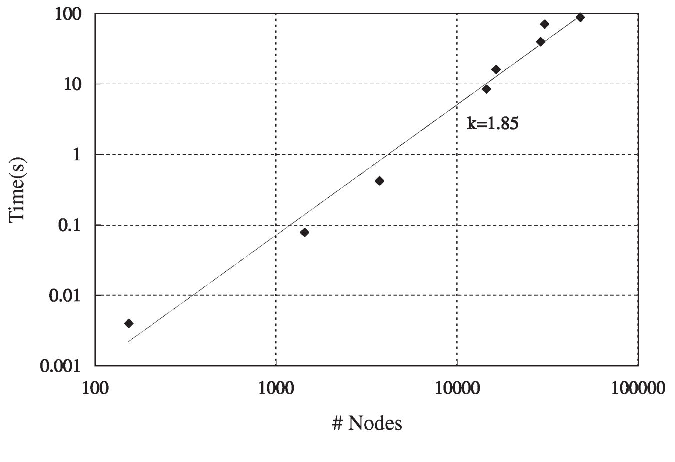


Fig. 6. Running time versus the number of the nodes for small antenna ratio bounds.

TABLE III
RESULTS OF JUMPER INSERTION WITH OBSTACLES

	a100 ratio=700		a1000 ratio=220		a10000 ratio=70		a20000 ratio=50	
% ob	# cuts	time(s)	# cuts	time(s)	# cuts	time(s)	# cuts	time(s)
0%	18	0.002	54	0.016	376	0.514	444	1.969
10%	22	0.002	56	0.014	394	0.473	464	1.850
20%	–	0.002	56	0.014	421	0.461	483	1.756
30%	–	0.002	60	0.016	438	0.455	508	1.617
40%	–	0.002	63	0.014	485	0.371	531	1.420
50%	–	<1ms	69	0.016	532	0.320	574	1.223
60%	–	<1ms	87	0.012	625	0.299	618	1.004
70%	–	<1ms	–	0.010	979	0.236	717	0.775
80%	–	<1ms	–	0.008	–	0.156	976	0.615
90%	–	<1ms	–	0.008	–	0.105	–	0.285

	n2676 ratio=30		n12052 ratio=20		n22373 ratio=55		n34728 ratio=26	
% ob	# cuts	time(s)	# cuts	time(s)	# cuts	time(s)	# cuts	time(s)
0%	6	0.051	23	0.799	359	2.086	6	4.914
10%	6	0.055	25	0.707	408	1.885	10	4.633
20%	6	0.051	65	0.635	460	1.672	13	4.068
30%	6	0.047	69	0.576	518	1.455	13	3.598
40%	15	0.043	118	0.523	623	1.318	20	2.955
50%	15	0.041	159	0.426	751	1.053	20	2.406
60%	19	0.037	183	0.371	881	0.834	20	2.025
70%	29	0.029	226	0.293	1207	0.650	24	1.529
80%	44	0.029	266	0.225	–	0.400	32	1.029
90%	–	0.018	–	0.139	–	0.236	50	0.590

We then run the RatioPart algorithm with obstacles. We randomly choose edges to be forbidden, which means that no jumper insertion is allowed on them. For each benchmark, we choose one representative bound, which is the second largest one among the six bounds in Table II. Scenarios with different percentages of forbidden edges are tested. The results are reported in Table III. The column "% ob" shows the percentage of the forbidden edges. The cells with "–" denote that no valid partitioning under the upper bound can be found with the obstacles presented. Note that the obstacle setting here is only for simplicity while the RatioPart algorithm can handle more general settings, e.g., the one in the work of Su *et al.* [13] where the obstacles can forbid inserting jumpers on parts of an edge.

VI. Conclusion

In this paper, we presented an optimal algorithm for antenna avoidance via jumper insertion under the upper bound of the antenna ratio. The algorithm is based on the dynamic programming on free trees. The experimental results confirmed the effectiveness of our approach. Future works include routing with antenna planning and heuristics for practical running time reduction.

References

[1] R. H. J. M. Otten, R. Camposano, and P. R. Groeneveld, "Design automation for deepsubmicron: Present and future," in *Proc. DATE*, 2002, pp. 650–657.

[2] H. K.-S. Leung, "Advanced routing in changing technology landscape," in *Proc. Int. Symp. Phys. Des.*, 2003, pp. 118–121.

[3] H. C. Shin and C. Hu, "Thin gate oxide damage due to plasma processing," *Semicond. Sci. Technol.*, vol. 11, no. 4, pp. 463–473, 1996.

[4] R. Rakkhit, F. P. Heiler, P. Fang, and C. Sander, "Process induced oxide damage and its implications to device reliability of submicron transistors," in *Proc. IEEE 38th Annu. Int. Rel. Phys. Symp.*, 1993, pp. 293–296.

[5] H. Watanabe, J. Komori, K. Higashitani, M. Sekine, and H. Koyama, "A wafer level monitoring method for plasma-charging damage using antenna PMOSFET test structure," *IEEE Trans. Semicond. Manuf.*, vol. 10, no. 2, pp. 228–232, May 1997.

[6] "Process-Induced Damage Rules (otherwise known as 'Antenna Rules')—General Requirements," in *MOSIS CMP and Antenna Rules*, Marina del Rey, CA: The MOSIS Service. [Online]. Available: http://www.mosis.org/Technical/Designrules/guidelines.html#antenna

[7] H. Shirota, T. Sadakane, M. Terai, and K. Okazaki, "A new router for reducing "antenna effect" in ASIC design," in *Proc. IEEE Custom Integr. Circuits Conf.*, 1998, pp. 601–604.

[8] P. H. Chen, S. Malkani, C. Peng, and J. Lin, "Fixing antenna problem by dynamic diode dropping and jumper insertion," in *Proc. Int. Symp. Quality Electron. Des.*, 2000, pp. 275–282.

[9] L. Huang, X. Tang, H. Xiang, D. Wong, and I. Liu, "A polynomial time optimal diode insertion/routing algorithm for fixing antenna problem," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 1, pp. 141–147, Jan. 2004.

[10] T. Y. Ho, Y. W. Chang, and S. J. Chen, "Multilevel routing with jumper insertion for antenna avoidance," *Integr.: VLSI J.*, vol. 29, no. 4, pp. 420–432, Jul. 2006.

[11] D. Wu, J. Hu, and R. Mahapatra, "Antenna avoidance in layer assignment," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 4, pp. 734–738, Apr. 2006.

[12] B. Y. Su and Y. W. Chang, "An exact jumper insertion algorithm for antenna effect avoidance/fixing," in *Proc. Des. Autom. Conf.*, 2005, pp. 325–328.

[13] B. Y. Su, Y. W. Chang, and J. Hu, "An optimal jumper insertion algorithm for antenna avoidance/fixing on general routing trees with obstacles," in *Proc. Int. Symp. Phys. Des.*, 2006, pp. 56–63.

[14] J. Wang and H. Zhou, "Optimal jumper insertion for antenna avoidance under ratio upper-bound," in *Proc. Des. Autom. Conf.*, 2006, pp. 761–766.

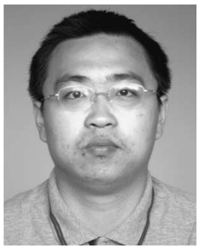
[15] H. Zhou, "Efficient steiner tree construction based on spanning graphs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 5, pp. 704–710, May 2004.

[16] A. B. Kahng, I. I. Mandoiu, and A. Zelikovsky, "Highly scalable algorithms for rectilinear and octilinear steiner trees," in *Proc. Asia South Pac. Des. Autom. Conf.*, 2003, pp. 827–833.

[17] L. J. Stockmeyer, "Optimal orientations of cells in slicing floorplan designs," *Inf. Control*, vol. 57, no. 2/3, pp. 91–101, May/Jun. 1983.

[18] L. P. P. P. van Ginneken, "Buffer placement in Distributed RC-tree network for minimal elmore delay," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1990, pp. 865–868.

[19] T. H. Cormen, C. E. Leiserson, R. H. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press, 2001.



Jia Wang received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 2002. He is currently working toward the Ph.D. degree in computer engineering at Northwestern University, Evanston, IL.

His research interests are on very large scale integrated computer-aided design, especially algorithm design.

Hai Zhou (SM'04) received the B.S. and M.S. degrees in computer science and technology from Tsinghua University, Beijing, China, in 1992 and 1994, respectively, and the Ph.D. degree in computer sciences from University of Texas, Austin, in 1999.

He is an Assistant Professor of electrical engineering and computer science with Northwestern University, Evanston, IL. Before he joined the faculty of Northwestern University, he was with the Advanced Technology Group, Synopsys, Inc. His research interests include very large scale integrated computer-aided design, algorithm design, and formal methods.

Dr. Zhou served on the technical program committees of many conferences on very large scale integration (VLSI) and computer-aided design such as ACM Design Automation Conference and IEEE International Conference on Computer-Aided Design. He was a recipient of the CAREER Award from the National Science Foundation in 2003.