Optimal Jumper Insertion for Antenna Avoidance under Ratio Upper-Bound *

Jia Wang and Hai Zhou
Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208, USA

ABSTRACT

Antenna effect may damage gate oxides during 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 works 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 non-blocked position on a wire for a jumper.

Categories and Subject Descriptors: J.6 [Computer-Aided Engineering]: Computer-Aided Design
General Terms: Algorithms
Keywords: Antenna Effect, Jumper Insertion

1. INTRODUCTION

Antenna effect is a phenomenon in VLSI fabrication where current caused by plasma process flows through 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 VLSI feature sizes. The relationship between the amount of damage and the antenna ratio has been studied for a long time [3, 4, 5]. Generally speaking, a wire segment acting as antenna may collect charging current when exposed to plasma. If the segment only connects to gate oxides but not diffusions, a voltage potential may build up and a discharging current tunneling through the gate oxides may 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 threshold voltages with spatial correlations. Besides the processing parameters, which are constant, the antenna ratio of total exposed antenna area to total gate oxide area determines the amount of voltage potential and thus the damage: smaller antenna ratio results in less damage. Antenna rules are commonly enforced as upper-bounds on the antenna ratio in design rules [6].

Fixing antenna problem after placement and routing stage is feasible and effective [7, 8, 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 [1, 2], correction after placement and routing may not be effective or even possible beyond the 0.13μ technology. The main reason is that with the scaling-down of the VLSI feature sizes, 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. So, considering antenna effect in an earlier stage and planning ahead is a must to avoid the problems.

Two recent works [10, 11] considered the antenna effect during the routing stage by combining jumper insertion to routing. In these approaches, jumper insertion is more like a process for antenna avoidance than antenna fixing in post-processing approaches. 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. However, none of the four works directly addressed the most important rule for the antenna effect – the upper-bound on antenna ratio. They only controlled the upper-bound on the total exposed antenna area connected to gate oxides. Such a simplification is not accurate for antenna-effect-aware routing. Since there may be multiple gate oxides connected to one wire segment, using a small upper-bound on the total exposed antenna area would over-constrain the routing. On the other hand, 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. Moreover, the situations could be worse since gates are commonly sized for performance and power consumption during design, which cannot be considered in previous research.

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 better result for antenna avoidance. We first formulate the RatioJI problem that models 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

*This work is supported by the NSF under award CCR–0238484.

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

in the routing tree and α is a factor depending on how to find a non-blocked 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 [15, 16], 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 2, backgrounds on antenna effect are introduced. The RatioJI problem is formulated in Section 3. Algorithm to solve the problem is presented in Section 4. After experimental results given in Section 5, Section 6 concludes the paper.

2. ANTENNA EFFECT

A detailed overview of the antenna effect can be found in the work [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, 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 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. Both the charging current density J_p and the tunneling current density J_{FN} are the functions of V_g when the technology parameters are given as constants. The following equation shows the relationship between the antenna ratio and the current densities:

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

As plotted in Figure 12 of the work [3], the charging current decreases with the increase of V_g; but the tunnelings 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 [7], there are three types. The first are the conductor layer pattern etching processes where the perimeters of the wires are exposed. So the exposed area is computed as the perimeter length of conductor layer patterns. The second are the ashing processes where the area of the wires are exposed. Thus the exposed area is the area of the conductor layer patterns. The third are the contact etching processes where the area of the contacts on the lower conductor layer are exposed. Therefore the exposed area is the total area of the contacts. Our problem formulation is general enough to handle all these types.

3. PROBLEM FORMULATION

Considering 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 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 2 jumpers are enough to satisfy the antenna rule. Thus c_e is written as

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

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 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 $\frac{l(S)}{g(S)} \leq R$. Note that it is possible to have no valid partitioning because of the obstacles.

Since inserting jumpers will degrade 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 ratio upper-bound is formulated as the following RatioJI problem.

PROBLEM 1 (RatioJI). *Suppose $T = (V, E)$ 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 $\frac{l(S)}{g(S)} \leq R$.*

4. OPTIMAL JUMPER INSERTION

4.1 Algorithm Overview

We solve the RatioJI problem by dynamic programming. Classical dynamic programming algorithms usually work on a rooted tree [15, 16]. 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 Figure 1. Starting from Figure 1 (a), there are three possible edges to be processed next as shown in (b), (c), and (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.

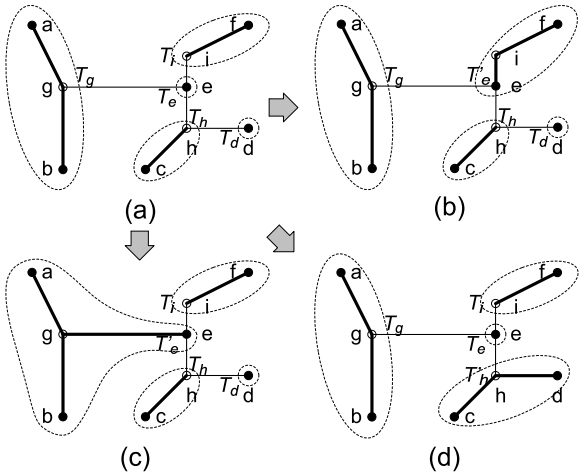

Figure 1: Process one edge by combining two trees. Bold edges are processed; others are unprocessed. Solid nodes are gates; others are Steiner points. In (a), the edges (a, g), (b, g), (f, i), and (c, h) are processed while the dominant partial solutions are maintained on trees T_g, T_d, T_h, and T_i.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

4.2 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 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 $\frac{l(S)}{g(S)} \leq R$. 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 can be 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.

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 is 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.*

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.

4.3 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 Lemma 2.

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 $l(p_2, v)$.*

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 Figure 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 back-tracking which is used to construct the optimal partitioning later.

Subroutine Combine
Inputs $e = (u, v)$, D_v, and D_u.
Outputs Updated D_v.

```
1  D* ← ∅.
2  For each (d_v, d_u) in the set D_v × D_u:
3  |  (n_v, x_v, I_v, B_v) ← d_v; (n_u, x_u, I_u, B_u) ← d_u.
4  |  For each case in Lemma 2:
5  |  |  Compute (n, x, I) and c_e.
6  |  |  B ← B_v ∪ {(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 ← D*.
```

Figure 2: The Combine subroutine.

In Figure 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 n's and I's 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 non-dominant 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.

4.4 The RatioPart Algorithm

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 ← {(0, 0, 0, ∅)}.
   |  Else:
5  |  |  D_u ← {(0, −g(u)R, 1, ∅)}.
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 ← the last v in the While loop.
11 d_1 ← the element with the least n in D_w
   whose I = 0.
12 d_2 ← the element with the least n in D_w
   whose I = 1 and x ≤ 0.
13 If none of d_1 and d_2 exists:
14 |  Report there is no valid partitioning.
   Else:
15 |  (n, x, I, B) ← the one with the smaller n.
16 |  ReportPartition (B).
```

Figure 3: The RatioPart algorithm.

Figure 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_u's are built on line 2 to 5. The **While** loop on line 6 to 9 processes the edges and updates the dominant partial solutions. The invariant of the loop is stated in Lemma 3.

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

The code on line 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 Lemma 4.

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

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 ReportPartition , as shown in Figure 4, takes B as the parameter and recursively reports the optimal partitioning. The search on line 3 of the ReportPartition subroutine takes 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.

Subroutine ReportPartition
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  |  ReportPartition (B_u).
```

Figure 4: The ReportPart subroutine.

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 line 6 to 9 terminates because the number of unprocessed edges is limited and decreases by one each time. The ReportPartition 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 $I = 1 \wedge x \le 0$. 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) [17] 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 Figure 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 are (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 Figure 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.

4.5 Complexity of the RatioPart Algorithm

The space complexity of the RatioPart algorithm is stated in Theorem 2.

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. So D_u requires at most $O(m_u|V|)$ storage and the total storage required by all the D_u's is

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

For the Combine subroutine, D^* is an array of at most $O(|V|)$ elements and every element requires at most $O(|V|)$ storage. So it requires at most $O(|V|^2)$ storage. For the ReportPartition 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 $O(|V|^2)$. □

To evaluate the time complexity, we assume that both functions g and l take constant time to compute. The time complexity is stated in Theorem 3.

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 non-blocked position on a wire for a jumper.*

PROOF.Consider the Combine subroutine. Suppose the 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 non-blocked 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 line 2 to 7 takes $O(\alpha N_u N_v)$ time. As discussed in Section 4.3, line 1 and 8 take $O(N_u + N_v)$ time. So 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. Denote 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 $\frac{A}{2}\alpha|V|$ and finally the potential is $\frac{A}{2}\alpha|V|^2$, the Combine subroutine takes at most

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

time when it is used in the RatioPart algorithm.

In the ReportPartition 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. So the time complexity is $O(\alpha|V|^2)$. □

5. EXPERIMENTS

The experiments are conducted on a Linux workstation with dual 933MHz Pentium III processors and 512MB memory where the RatioPart algorithm is implemented in C++ and compiled with GCC 3.4.3. Since there is no previous work considering the upper-bound of the antenna ratio, we do not compare our results to other's.

We generate 4 benchmarks containing 100, 1000, 10000, and 20000 gates respectively. The gates are randomly located in a 10000 by 10000 grid and a Steiner tree is constructed as the routing tree using the algorithm in the work [14]. All the gate sizes are assumed to be 1 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 1. The columns "name" and "# gates" show the name and the number of the gates of each benchmark respectively. 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 cut is needed.

Table 1: 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

We first run the RatioPart algorithm without obstacles. For each benchmark, 6 upper-bounds of the antenna ratio are tested and the results are reported in Table 2. 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. Moreover, in Figure 5, we plot the number of jumpers inserted and the running time in seconds vs. different upper-bounds for the benchmark a10000.

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 ratio upper-bound. Scenarios with different percentage of forbidden edges are tested. The results are reported in Table 3. 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

Table 2: 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

Table 3: 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

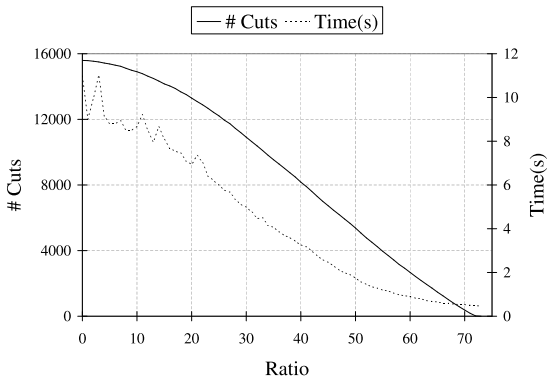

Figure 5: The number of the cuts and the running time vs. the antenna ratio upper-bound for a10000 without obstacles.

handle more general settings, e.g. the one in the work [13] where the obstacles can forbid inserting jumpers on parts of an edge.

6. 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 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.

7. REFERENCES

[1] R. H.J.M. Otten, R. Camposano, and P. R. Groeneveld. *Design Automation for Deepsubmicron: present and future.* In DATE, pages 650-657, 2002.

[2] H. K.-S. Leung. *Advanced Routing in Changing Technology Landscape.* In ISPD, pages 118-121, 2003.

[3] H C Shin and C. Hu. *Thin Gate Oxide Damage Due to Plasma Processing.* Semicond. Sci. Technol., vol. 11, pages 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 IEEE 38th Annu. Int. Reliability Phys. Symp., pages 293-296, 1993.

[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. Semiconductor Manufacturing, vol. 10, no. 2, pages 228-232, May 1997.

[6] *Process-Induced Damage Rules (otherwise known as "Antenna Rules") - General Requirements.* 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 IEEE Custom Integrated Circuits Conf., pages 601-604, 1998.

[8] P. H. Chen, S. Malkani, C. Peng, and J. Lin. *Fixing Antenna Problem by Dynamic Diode Dropping and Jumper Insertion.* In ISQED, pages 275-282, 2000.

[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. Computer-Aided Design, vol. 23, no. 1, pages 141-147, Jan 2004.

[10] T. Y. Ho, Y. W. Chang, and S. J. Chen. *Multilevel Routing with Antenna Avoidance.* In ISPD, pages 34-40, 2004.

[11] D. Wu, J. Hu, and R. Mahapatra. *Coupling Aware Timing Optimization and Antenna Avoidance in Layer Assignment.* In ISPD, pages 20-27, 2005.

[12] B. Y. Su and Y. W. Chang. *An Exact Jumper Insertion Algorithm for Antenna Effect Avoidance/Fixing.* In DAC, pages 325-328, 2005.

[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 ISPD, pages 56-63, 2006.

[14] H. Zhou. *Efficient Steiner Tree Construction Based on Spanning Graphs.* IEEE Trans. Computer-Aided Design, vol. 23, no. 5, pages 704-710, May 2004.

[15] L. J. Stockmeyer. *Optimal Orientations of Cells in Slicing Floorplan Designs.* Information and Control, vol. 57, no.2-3, pages 91-101, May/June 1983.

[16] L. P.P.P. van Ginneken. *Buffer Placement in Distributed RC-tree Network for Minimal Elmore Delay.* In Proc. IEEE Int. Symp. Circuits Syst., pages 865-868, 1990.

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