Strongly NP-hard Discrete Gate-Sizing Problems

Wing Ning Li, Member, IEEE

Abstract—The discrete gate-sizing problem has been studied by several researchers recently. Some complexity results have been obtained, and a number of heuristic algorithms have been proposed. For circuit networks that are restricted to the set of trees, or series-parallel graphs, pseudo-polynomial time algorithms to obtain the exact solution have also been proposed, though none can be extended to circuit networks that are arbitrary directed acyclic graphs (dags). We prove that the problem is strongly NP-hard. Our result implies that for arbitrary dags, there is no pseudo-polynomial time algorithm to obtain the exact solution unless \( P=NP \). We also prove that the absolute approximation discrete gate sizing problem is strongly NP-hard. These results provide insight into the difficulties of the problem and may lead to better heuristics.

Index Terms—Discrete gate sizing, timing optimization, technology mapping, NP-hard, cell selection, complexity.

I. INTRODUCTION

Two well known applications of gate sizing in circuit design are in the area of timing optimization and technology mapping. In the case of timing optimization, the current implementation (or binding) of the circuit modules in a combinational network cannot deliver the performance (speed) that is required, and the sizes of many modules must be increased in order to meet the timing requirement, i.e., all path delays \( \leq T_{\text{Max}} \). Consequently, the area of the circuit is also increased, which is an undesirable effect that should be minimized. An interesting and challenging problem is how to keep this increase in area to a minimum and still meet the timing requirement. Similarly, in the case of technology mapping, a combinational network that has not been implemented and a set of timing requirements (typically \( T_{\text{Max}} \)) are given, and an implementation (or binding) of each circuit module in the network is sought to meet the timing requirements and at the same time minimize the area of the resulting circuit.

There are two formulations of the gate-sizing problem. One is the continuous sizing, where each gate size is allowed to vary continuously between its minimum and maximum sizes. This formulation is more suitable for transistor sizing in a custom or semi-custom design environment. The continuous gate sizing problem has been studied by a number of groups [2], [3], [5], [10]. The other formulation of the gate-sizing problem is the discrete sizing, where each gate size is allowed to vary among its finite set of predetermined values. Typically, a precharacterized library of cells provides these predetermined values. This formulation is more suitable for transistor sizing or cell selection in a design environment that has the support of a cell library. The discrete gate sizing problem has been studied by a number of groups [1], [6]–[8]. In these studies, the delay model used is fanout independent; the delay of each cell is a constant regardless of what it drives. This delay model is a simplification of the delay model considered in [9], [11]: the delay of each cell is a constant (intrinsic delay) plus another constant (drive resistance) times the total load. In this paper, as in [1], [6]–[8], the fanout independent delay model is used. The computational complexity results for this simpler delay model may continue to hold for the more complicated delay models, as the fanout independent delay model is a special case of the more general delay models.

In [1] and [6], the complexity of the discrete gate sizing problem has also been studied. Chan [1] has shown that the discrete gate sizing problem with minimum and maximum delay constraints is NP-hard for networks with tree topology, and he has obtained a pseudo-polynomial time algorithm to obtain a minimum area solution for networks with this topology. Li et al. [6] have shown a stronger result that the discrete gate sizing problem with only maximum delay constraint is NP-hard for networks with chain topology, and they have obtained a pseudo-polynomial time algorithm to obtain a minimum area solution for networks with series-parallel graph topology. Both pseudo-polynomial time algorithms, however, cannot be extended to obtain a minimum area solution for networks with arbitrary topology. Whether such a pseudo-polynomial time algorithm exists or not is an interesting question and has not been known. To answer such an open question, we prove that the discrete gate sizing problem in general (any network topology is allowed) is NP-complete in the strong sense [4] in this paper. This result implies that there is no pseudo-polynomial time algorithm for discrete gate sizing problem unless \( P=NP \).

The rest of this paper is organized into three sections. In Section II, we define the terminology and notation, and we formulate and formally define the discrete gate sizing problem. In Section III, we present the complexity results of the problem. In the last section, we present our concluding remark and future research into the problem.

II. TERMINOLOGY AND NOTATION

The structure of a combinational logic network can be represented by a dag \( G = (V, E) \) where \( V = \{v_1, v_2, \cdots, v_l\} \), and \( E \subset V \times V \). Each gate \( g \) of the network is represented by a node \( v_i \) in \( G \), and each connection from the output of gate \( g \) to the input of gate \( g_i \) is represented by a directed edge \( (v_i, v_j) \). Those nodes with zero in-degree in \( G \) correspond to the primary inputs of the network, and nodes with zero out-degree correspond to the primary outputs of the network. Since we are only interested in those paths that start from a primary input and terminate at a primary output, in the remaining of this paper a path that starts from a zero in-degree node and terminates at a zero out-degree node is defined as a path. There is a type function, \( f \), which maps each node \( v \) to the gate type of \( g \) (a gate type is referred to by an integer index). Thus, a combinational logic network can be specified by a dag \( G \) and a type function \( f \).

In this discussion, a combinational logic circuit network is often referred to by its dag \( G \), with its type function being implicitly understood. The words network, circuit, and dag are used interchangeably, as well as the words gate, module, and node.

A cell library can be represented by a family of finite sets, \( L_1, L_2, \cdots, L_K \), where \( L_i, 1 \leq i \leq K \), contains one or more logically equivalent cells that realize gate type \( i \), and \( K \) is the total number of gate types that the cell library supports. In addition to its internal design, each cell is characterized by its area and delay (power can be dealt with similarly). Since the gate-sizing problem focuses only on the area and delay aspects of the circuit,
each cell is represented by a pair of numbers (area, delay), and 
$L_j = \{(a_{1j}, d_{1j}), (a_{2j}, d_{2j}), \ldots, (a_{pj}, d_{pj})\}$ is a finite set of pairs, where $p_i$ is a positive integer and stands for the total number of cells that realize gate type $i$, $a_{ij}$ is the area of the $j$th cell that realizes gate type $i$, and $d_{ij}$ is the delay of that cell. 

Given a circuit network, described by $G$ and $f$, and a cell library $L_1, L_2, \ldots, L_K$, an implementation of the network by the cell library is a binding of each node $v$ in $G$ to one of the cells in $L_f(v)$. Let $S = (s_1, s_2, \ldots, s_n)$ be an implementation, where $1 \leq s_i \leq |L_f(v_i)|$. The area of the circuit under the implementation $S$ is denoted by 

$$A(S, G) = \sum_{1 \leq i \leq n} a_{f(v_i), s_i}.$$ 

Let $v_{i1}, v_{i2}, \ldots, v_{in}$ be a path, $p$, in $G$. The delay of this path under implementation $S$ is denoted by 

$$d(S, p) = \sum_{1 \leq i \leq t} d_{f(v_i), s_i}.$$ 

The delay of the circuit under implementation $S$ is, denoted by $D(S, G)$, 

$$\text{Max}\{d(S, p) | p \text{ is a path in } G\}.$$ 

We now define the decision version of the discrete gate sizing problem formally as:

**DGS** [Discrete Gate Sizing].

*Input:* A dag $G$, a type function $f$, a cell library $L_1, L_2, \ldots, L_K$, and, two numbers $A_{Max}$ and $T_{Max}$.

*Output:* "Yes" iff there is an implementation of $G, S$, by the cell library $L_1, L_2, \ldots, L_K$ such that $A(S, G) \leq A_{Max}$ and $D(S, G) \leq T_{Max}$.

The optimization version of DGS, ODGS, seeks for an implementation, $S$, such that $A(S, G)$ is minimum among all possible implementations under which the delay of the circuit is no more than $T_{Max}$. Other optimization variations such as those described in [3] can be defined similarly.

### III. Complexity Results

To establish that the discrete gate-sizing problem is NP-complete in the strong sense, we shall make use of the known NP-complete problem 3SAT [4], which is defined as the following.

*Input:* A collection of $m$ clauses $C_1, C_2, \ldots, C_m$ over $n$ variables $x_1, x_2, \ldots, x_n$ such that each clause $C_i = (l_{i1} + l_{i2} + l_{i3})$, $1 \leq i \leq m$, is the disjunction of exactly three literals (variables or their negations).

*Output:* "Yes" iff there is a truth assignment to the variables such that each clause is true.

**Overview of the construction:** To reduce 3SAT to DGS, we first construct a subnetwork for each variable and a subnetwork for each clause, then we connect the subnetworks for the set of variables to the subnetworks for the set of clauses to obtain the dag $G$. A type function and a cell library are defined (constructed) next. Depending on how the variable subnetwork of $x_i$ is implemented, two possible truth values of $x_i$ can be interpreted. Under this interpretation, a clause subnetwork is implementable with respect to the delay and area constraints if the implementation of its corresponding variable subnetworks leads to one of its literals to true. The overall construction is designed in such a way that the 3SAT instance is satisfiable if the DGS instance is implementable with area under $A_{Max} = 10(n + m)$ and delay under $T_{Max} = 11$.

It should be pointed out to the reader that in our construction, the cell library and the $T_{Max}$ value are independent to the 3SAT instance. This independence to the 3SAT instance will not compromise our complexity results; instead, it provides us with stronger complexity results. Note that our construction shows that a restricted sub-problem of DGS is strongly NP-hard, which trivially implies DGS is strongly NP-hard.

**A. Variable Subnetwork**

For each variable $x_i$, $1 \leq i \leq n$, a subnetwork shown in Fig. 1 is built. Fig. 1(a) shows the network with the names displayed, and Fig. 1(b) shows the same network with the delays displayed. There are two kinds of gates in the subnetwork represented, respectively, by boxes and circles. We shall call them box-gate and circle-gate.

For each box-gate, there is only one cell in the cell library to realize that gate, and the question of which set of cells to be selected from the cell library to implement those box-gates can be trivially answered, since we do not have a choice here. The number shown inside each box represents the delay value of the cell that realizes the gate, and the area of the cell is always 1. For each circle-gate, unfortunately, there are two cells in the cell library to realize that gate, and a choice must be made as to which cell is selected to implement the circle-gate. All the circle-gates are of the same type, say 1, and $L_1 = \{(1, 4), (2, 1)\}$. Let cell1 be the first pair, and cell2 the second one.
In the subnetwork shown in Fig. 1(a), nodes $x_1, x_2,$ and $v_{1,1}$ are the primary inputs, and $v_{1,5}, v_{1,6}$ the primary outputs. There are a total of five different paths. Let $T_{Max} = 11$. Then under any implementation that satisfies this $T_{Max}$ value, the minimum area of the subnetwork is 10. Although the minimum possible area of 8 can be achieved by the implementation that binds cell1 to all the circle gates, this makes four paths with delay that is greater than $T_{Max}$. To achieve an area of 9, an implementation must have three of the four circle gates bound to cell1. The three possible implementations with area 10 and delay $T_{Max}$ = 11 are shown in Fig. 2. The implementation in Fig. 2(a) corresponds to $x_1$ being true, in Fig. 2(b) to $x_1$ being false, and in Fig. 2(c) to $x_1$ being don’t care.

**B. Clause Subnetwork**

For each clause $C_i, 1 \leq i \leq m$, a subnetwork shown in Fig. 3 is built. Boxes and circles in Fig. 3 carry the same meaning as those shown in Fig. 1. The type of all the circle-gates in Fig. 3 is the same as that of circle-gates in Fig. 1.

In this subnetwork, $l_{i,2}, l_{i,3},$ and $c_{i,1}$ are primary inputs, and $c_{i,3}, c_{i,4},$ and $c_{i,5}$ are primary outputs. Since each of the gates $l_{i,1}, l_{i,2},$ and $l_{i,3}$ receives an input from the corresponding variable subnetwork, $l_{i,1}$ is also considered as a primary input for the subnetwork (for path consideration). Thus, there are a total of nine different paths in this subnetwork. Let the input delay of a gate be the longest path delay from the primary inputs of the circuit network to that gate. Then the input delay of gate $l_{i,j}, 1 \leq j \leq 3$, is either 1 or 4 depending on how its corresponding variable subnetwork is implemented.

Again, let us consider $T_{Max} = 11$. To satisfy the $T_{Max}$ constraint, the minimum possible area of the subnetwork under any implementation is 10. The area of the box-gates is 5, and that of the circle-gates is 5. To have the $c_{i,1}, l_{i,1}, c_{i,2}, c_{i,3}$ path delay no more than $T_{Max}$, two of the three circle-gates must be bound to cell2, and the remaining one is bound to cell1, resulting a total area of 5.

This minimum possible area can be achieved if at least one of the input delays of $l_{i,1}, l_{i,2}, l_{i,3}$ is 1. Fig. 4 shows the three implementations for the three cases in which exactly one of the input delays is 1. Only the gate delays of the resulting circuits are shown here, and the area of each of the resulting circuits is this minimum possible area of 10. These implementations can also be used to achieve the minimum area for those cases in which more than one input delays are 1. However, if all input delays of $l_{i,1}, l_{i,2}, l_{i,3}$ are 4, then to have all path delays under $T_{Max}$, $l_{i,1}, c_{i,2},$ and $c_{i,3}$ must be bound to cell2. The area of the subnetwork under this implementation is 11.

Now we are ready to describe how the variable subnetworks are connected to the clause subnetworks, and show that $3SAT$ reduces to $DGS$. Fig. 5 shows the dag after interconnecting all the subnetworks that corresponds to the $3SAT$ instance: $C_1 = (x_1 + x_2 + x_3), C_2 = (x_1 + x_2 + x_3), C_3 = (x_1 + x_2 + x_3)$. In this example, $A_{Max} = 60 and T_{Max} = 11$.

**Lemma 1**: $3SAT$ reduces to $DGS$.

**Proof**: Let $x_1, x_2, \ldots, x_n$ be the variables and $C_1, C_2, \ldots, C_m$ be the clauses of a $3SAT$ instance, where the three literals in clause $C_i$ are denoted by $l_{i,1}, l_{i,2}, l_{i,3}$. We shall show how to construct, in polynomial time, an instance of $DGS$ for which the output is “yes” if that for the $3SAT$ instance is also “yes.”

**Construction 1**: The dag is constructed as shown below.

1) For each variable $x_i$, define:

$$V_i' = \{x_i, \overline{x_i}, v_{i,1}, v_{i,2}, v_{i,3}, v_{i,4}, v_{i,5}, v_{i,6}\}$$
For each clause \( C_i \), define:

\[
E_{i'} = \{ (c_{i1}, c_{i2}), (c_{i2}, c_{i3}), (c_{i3}, c_{i1}) \}
\]

Let \( V' = U_{i \leq n} V_i \), and \( E' = U_{i \leq n} E_i' \). Note \( G_i' = (V_i', E_i') \) is the subnetwork shown in Fig. 1(a).

2) For each clause \( C_i \), define:

\[
V_i' = \{ z_{12}, z_{13}, z_{23}, \ldots, z_{n-1,n} \}
\]

\[
E_i' = \{ (z_{12}, z_{23}), (z_{23}, z_{34}), \ldots, (z_{n-1,n}, z_{n-2,n-1}) \}
\]

Let \( V'' = U_{i \leq n} V_i' \), and \( E'' = U_{i \leq n} E_i'' \). Note \( G_i'' = (V_i'', E_i'') \) is the subnetwork shown in Fig. 3(a).

3) Define:

\[
E_0 = \{ (x_i, z_{12}) | 1 \leq i \leq n \}
\]

Let \( E_1' = E_0 \cup E_1 \). \( E'' \) is the set of connections from the set of variable subnetworks to the set of clause subnetworks.

4) Let \( V'' = V'' \cup V'' \), and \( E' = E' \cup E'' \cup E''' \). \( G = (V, E) \) is the dag of the constructed DGS instance.

Next we define (construct) the type function \( f \) as:

\[
f(x) = \begin{cases} 
1 & \text{if } x \in T_1 \\
2 & \text{if } x \in T_2 \\
3 & \text{if } x \in T_3 \\
4 & \text{if } x \in T_4 
\end{cases}
\]

where

\[
T_1 = \{ x_i | 1 \leq i \leq n \} \cup \{ x_i | 1 \leq i \leq n \} \cup \{ v_{ij} | 1 \leq i \leq n \} \cup \{ v_{ij} | 1 \leq i \leq n \} \cup \{ l_{i,j} | 1 \leq i \leq n \} \cup \{ l_{i,j} | 1 \leq i \leq n \} \cup \{ l_{i,j} | 1 \leq i \leq n \} \cup \{ c_{ij} | 1 \leq i \leq m \}
\]

\[
T_2 = \{ x_i, x_j | 1 \leq i \leq n \} \cup \{ v_{ij} | 1 \leq i \leq n \} \cup \{ l_{i,j} | 1 \leq i \leq n \} \cup \{ l_{i,j} | 1 \leq i \leq n \} \cup \{ l_{i,j} | 1 \leq i \leq n \} \cup \{ c_{ij} | 1 \leq i \leq m \}
\]

\[
T_3 = \{ c_{ij} | 1 \leq i \leq m \}
\]

\[
T_4 = \{ v_{ij} | 1 \leq i \leq n \} \cup \{ v_{ij} | 1 \leq i \leq n \} \cup \{ l_{i,j} | 1 \leq i \leq n \} \cup \{ l_{i,j} | 1 \leq i \leq n \} \cup \{ l_{i,j} | 1 \leq i \leq n \} \cup \{ c_{ij} | 1 \leq i \leq m \}
\]

For the cell library we have: \( L_1 = \{ (1, 4), (2, 1) \} \), \( L_2 = \{ (1, 2) \} \), \( L_3 = \{ (1, 3) \} \), \( L_4 = \{ (1, 4) \} \). Finally let \( A_{max} = 10(n + m) \), and \( T_{max} = 11 \).

If the 3SAT instance is satisfiable, then there is a truth assignment to \( x_1, x_2, \ldots, x_n \) such that \( C_1, C_2, \ldots, C_m \) are true. From this truth assignment, we derive an implementation of the constructed DGS instance as follows. If \( x_i \) is true in this truth assignment, then the implementation binds the subnetwork \( G_i \) as shown in Fig. 2(a). This binding ensures that the input delay of all those gates connected to \( x_i \) is 1. So when \( x_i \) is true, the implementation binds the subnetwork \( G_i \), as shown in Fig. 2(b). Again, when \( x_i \) is true, if the \( k \)th, \( 1 \leq k \leq 3 \), literal of clause \( C_i \), is \( x_i \), then the input delay of gate \( G_i \), is 1. If \( x_i \) is false in this truth assignment, then the implementation of the clause subnetworks binds the subnetwork \( G_i \), as shown in Fig. 2(a). Then, when \( x_i \) is false, if the \( k \)th, \( 1 \leq k \leq 3 \), literal of clause \( C_i \), is \( x_i \), then the input delay of gate \( G_i \), is 1. Thus, under this truth assignment, if clause \( C_i \) is true, then at least one of the input delays of the gates \( G_i \), is 1 in the clause subnetwork, \( G_i \), is 1.

In both the bindings shown in Fig. 2(a) and 2(b), the area of the subnetwork remains 10 and no path of the subnetwork has delay more than 11. The total area of the variable subnetworks under this implementation is 10n.

Since all the clauses are true under this truth assignment, then at least one of the literals in each clause is true. Based on which of the literals is true, the implementation of the clause subnetworks is constructed as the following: If the first literal \( l_{i,1} \) is true, then the input delay of \( l_{i,1} \) is 1 and the implementation binds the subnetwork \( G_i' \), as shown in Fig. 4(a). If the first literal is false...
and the second literal is true, then the implementation binds \( G''_i \), as shown in Fig. 4(b). Finally, if \( l_{1,3} \) is the only true literal, then the implementation binds \( G''_i \), as shown in Fig. 4(c). In the three bindings shown in Fig. 4, the area of the subnetwork remains 10 and no path going through the subnetwork has delay more than 11. The total area of the clause subnetworks under this implementation is 10m. So under this implementation, \( S \), we have \( A(S,G) = 10(n + m) \), and \( D(S,G) = 11 \). Hence, if the output to the 3SAT instance is "yes," the output to the constructed DGS instance is "yes."

Now suppose the output to the DGS instance is "yes." Then there is an implementation, \( S \), under which \( A(S,G) \leq 10(n + m) \) and \( D(S,G) \leq 11 \). Since this implementation guarantees that all the path delays are no more than 11, then under such an implementation the minimum possible area of each variable subnetwork is 10 and that of each clause subnetwork is also 10. This minimum possible area of 10 must be the area of each variable subnetwork and of each clause subnetwork, otherwise the condition of \( A(S,G) \leq 10(n + m) \) is violated. Therefore, under this implementation, one of the bindings shown in Fig. 2 is used for each variable subnetwork \( G'_i \), and one of the bindings shown in Fig. 4 is used for each clause subnetwork \( G''_i \).

Now we define the truth assignment as the following: If the bindings shown in Fig. 2(a) and Fig. 2(c) are used to implement \( G'_i \), set \( x_i \) to true; otherwise set \( x_i \) to false. Under this truth assignment, each clause must be true since one of its literals is true. Note that if the binding of Fig. 4(a) is used to implement \( G''_i \), then the first literal of \( G''_i \) is true in this truth assignment, if the binding of Fig. 4(b) is used the second literal is true, and if that of Fig. 4(c) is used the third literal is true.

Hence, the output for the 3SAT instance is "yes" if that of the constructed DGS is also "yes."

In the network instance constructed using Construction 1, there are three types of box-gates, and there are exactly three cells in the cell library, respectively, with delay 2, 3, and 4 (the area of each cell is 1), to realize gates of each type. By replacing each box-gate whose corresponding cell has delay \( k \), \( 2 \leq k \leq 4 \), by a chain of \( k \) identical box-gates whose corresponding cell has delay 1 (the area is also 1), we reduce the types of box-gate from 3 to 1, and the number of cells that realize box-gates in the cell library to 1. With this modification Lemma 1 can still be proved similarly if we adjust \( A_{\text{DGS}} \) to \( 18n + 20m \). Note that the network instance constructed using Construction 1 does not have any reconvergent path.

**Theorem 1:** DGS is strongly NP-complete even if each network contains only two types of gates with no reconvergent path, each cell library contains only three cells with one cell to realize the gates of one type, and the other two to realize that of the other type. Furthermore, each cell area is limited to 1 and 2, and delay to 1 and 4.

Proof: This follows from the obvious fact that DGS is in NP, the above discussion, and Lemma 1.

If the network instances are allowed to have reconvergent paths, then we can further restrict the delay value of each cell from 1 and 4 to 1 and 3, and DGS remains strongly NP-complete.

The reduction in which each cell delay is restricted to 1 and 3 is very similar to that of Lemma 1. In fact, the construction of each variable subnetwork is identical to that of Fig. 1. Since \( L_1 \) is changed from \( \{(1,4),(2,1)\} \) to \( \{(1,3),(2,1)\} \), \( T_{\text{DGS}} \) is adjusted from 11 to 9. For the new \( L_1 \) and \( T_{\text{DGS}} \), the property that under any implementation that ensures the delay of the subnetwork is no more than \( T_{\text{DGS}} \), the minimum area of the subnetwork is 10 continues to hold.

The three possible implementations of each variable subnetwork with area 10 and delay 9 are shown in Fig. 6.

Now the construction of each clause subnetwork is slightly different from that of Fig. 3. For each clause \( C_i \), \( 1 \leq i \leq m \), a subnetwork shown in Fig. 7 is built. Both dags shown in Fig. 3 and 7 look a lot alike. In fact, the dag of Fig. 7 is obtained from that of Fig. 3 by adding two box-gates (with delay 2) \( c_6 \) and \( c_7 \) along with four edges \( (c_{1,1},c_{6,1}), (c_{6,1},c_{2,1}), (l_{1,1},c_{1,1}), \) and \( (c_{7,1},c_{3,1}) \), and by changing the type of box-gate \( c_1 \) from 3 to 2 (i.e., changing the cell delay from 3 to 2). Again, gates \( c_{1,1}, l_{1,1}, c_{1,2}, \) and \( l_{1,2} \) are considered as the primary inputs of the subnetwork, and \( c_{1,3}, c_{1,4}, \) and \( c_2 \) as the primary outputs. Now there are a total of thirteen paths. Note that reconvergent paths have been introduced in the subnetwork.

Let us consider \( T_{\text{DGS}} = 9 \). To satisfy the \( T_{\text{DGS}} \) constraint, the minimum possible area of the subnetwork under any implementation is 12. The area of the box-gates is 7, and that of the circle-gates is 5. Note that in the previous clause subnetwork, the delay of a single path \( c_{1,1}, l_{1,1}, c_{1,2}, c_{1,3} \) to be under \( T_{\text{DGS}} = 11 \) is sufficient to force two of the three circle-gates to be bound to cell2 and the minimum area of circle-gates to be 5. For the new clause subnetwork, the delay of three paths to be under \( T_{\text{DGS}} = 9 \) is needed to achieve the same result. These paths are \( p_1 = c_{1,1}, l_{1,1}, c_{1,2}, c_{1,3} \) and \( p_2 = c_{1,1}, c_{1,2}, c_{2,3}, c_{3,1}, c_{3,2}, c_{3,3}, c_{1,1}, l_{1,1}, c_{1,2}, c_{1,3} \), each of which ensures, respectively, that gates \( l_{1,1} \) and \( c_{1,2}, c_{2,3} \) and \( c_{1,3} \) and \( l_{1,2} \) and \( c_{3,3} \) cannot be bound to cell1 simultaneously. So if one of the circle gates is bound to cell1, then the other two must be bound to cell2, and a minimum total area of 5 results.

Although the input delay of gates \( l_{1,1}, l_{1,2}, \) and \( l_{1,3} \) is either 1 or 3 now, the property that this minimum possible area can be achieved if at least one of the input delays of \( l_{1,1}, l_{1,2}, \) and \( l_{1,3} \) is 1, which holds for the previous clause subnetwork, still holds for the new clause subnetwork. Fig. 8 shows the three implementations of the new clause subnetwork corresponding to that of Fig. 4. However, if all input delays of \( l_{1,1}, l_{1,2}, l_{1,3} \) are 3, then we have all path delays under \( T_{\text{DGS}} = l_{1,1}, c_{1,2}, c_{1,3} \) must be bound to cell2. The area of the subnetwork under this implementation is 13.

Given a 3SAT instance as described in Lemma 1, this new construction is formally defined as:

**Construction 2:** The dag is constructed as shown below.

1) Same as part 1) of Construction 1.
Fig. 7. New clause subnetwork example.

2) For each clause $C$, define:

$V''_C = \{l_1, l_2, l_3, c_1, c_2, c_3, c_4, c_5, c_6, c_7\}$

$E''_C = \{(l_1, c_2), (l_1, c_3), (l_1, c_4), (l_2, c_2), (l_3, c_3), (c_1, c_2), (c_1, c_3), (c_1, c_4), (c_2, c_3), (c_2, c_4)\}$

Let $V'' = \cup_{C \leq C_m} V''_C$, and $E'' = \cup_{C \leq C_m} E''_C$. Note $G'' = (V'', E'')$ is the subnetwork shown in Fig. 7(a).

3) Same as part 3) of Construction 1.

4) Let $V = V' \cup V''$, and $E = E' \cup E'' \cup E'''$. $G = (V, E)$ is the dag of the constructed DGS instance.

Next we define (construct) the type function $f$ as:

$$f(x) = \begin{cases} 
1 & \text{if } x \in T_1 \\
2 & \text{if } x \in T_2 \\
4 & \text{if } x \in T_4
\end{cases}$$

where

$T_1 = \{x \mid 1 \leq i \leq n\} \cup \{\bar{x} \mid 1 \leq i \leq n\} \cup \{c_{ij} \mid 1 \leq i \leq m, 2 \leq j \leq 3\}$

$T_2 = \{v_{ij} \mid 1 \leq i \leq n\} \cup \{c_{ij} \mid 1 \leq i \leq m\} \cup \{\bar{v}_{ij} \mid 1 \leq i \leq m\}$

$T_4 = \{v_{ij} \mid 1 \leq i \leq n\} \cup \{c_{ij} \mid 1 \leq i \leq m, 5 \leq j \leq 7\}$

For the cell library we have:

$L_1 = \{(1, 3), (2, 1)\}$

$L_2 = \{(1, 2)\}$

$L_4 = \{(1, 4)\}$.

Finally let $A_{\text{max}} = 10n + 12m$, and $T_{\text{max}} = 9$.

**Theorem 2:** DGS is strongly NP-complete even if each network contains only two types of gates, each cell library contains only three cells with one cell to realize the gates of one type, and the other two to realize that of the other type. Furthermore, each cell area is limited to 1 and 2, and delay to 1 and 3.

**Proof:** This follows from using the arguments of Lemma 1 on the network instance built by Construction 2, the discussion that leads to Theorem 1, and the obvious fact that DGS is in NP.

Theorems 1 and 2 imply that ODGS is strongly NP-hard even for a very restricted set of problem instances. Since the problem of finding optimal solution to ODGS is strongly NP-hard, we turn our attention to approximation algorithms and heuristics. The $k$-absolute approximation DGS is defined as the following.

*Input:* A dag $G$, a type function $f$, a cell library $L_1, L_2, \ldots, L_K$, and $T_{\text{max}}$.

*Output:* An implementation, $S'$, of $G$ by the cell library $L_1, L_2, \ldots, L_K$ such that $|A(S', G) - A(S_{\text{opt}}, G)| \leq k$, where $S_{\text{opt}}$ is the ODGS output (solution) for the input instance.

**Theorem 3:** For every constant $k$ the $k$-absolute approximation DGS is strongly NP-hard even if each network contains only two types of gates and has no reconvergent path, and if each cell library contains only three cells with one cell to realize the gates of one type and the other two to realize that of the other type. Furthermore, each cell area is limited to 1 and 2, and delay to 1 and 4.

**Proof:** Without loss of generality, let us consider the constant $k$ as a positive integer. For the cases where $k$ is not an integer, by appropriately selecting the unit of area we can reduce them to that of an integer.

The network $G$, constructed using Construction 1 has the property that if the output for the 3SAT instance is “no,” then under any implementation of $G, S,$ which satisfies $D(S, G) \leq T_{\text{max}}, A(S, G)$ is at least one more than $10(n + m)$, the minimum area obtainable
IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 13, NO. 8, AUGUST 1994

only when the output for the corresponding 3SAT instance is "yes."

Now we construct a network, \( G' \), which consists solely of \( k + 1 \) copies of \( G \). With the type function, \( f' \), being defined accordingly and the same cell library and \( T_{\text{Max}} \) as in that of Lemma 1, we have a \( k \)-absolute approximation DGS instance. From the proof of Lemma 1, we have that the output for the 3SAT instance is "yes" if there is an implementation, \( S' \), of \( G' \) such that \( D(S', G') = T_{\text{Max}} \), and \( A(S', G') = (k + 1)1\{n + m\} \). On the other hand, if the output for the 3SAT instance is "no," then under any implementation of \( G', S' \) which satisfies \( D(S', G') \leq T_{\text{Max}}, A(S', G') \), is at least \( k + 1 \) more than \( (k + 1)1\{n + m\} \). In other words, if the output for the 3SAT instance is "no" then \( (k + 1)1\{n + m\} \) would be the minimum possible area of the circuit that any implementation under which the circuit delay is no more than \( k \). Hence, the 3SAT instance is satisfiable if \( A(S', G') \leq (k + 1)1\{n + m\} \), where \( S' \) is the output for the \( k \)-absolute approximation DGS instance constructed above.

Therefore, 3SAT reduces to \( k \)-absolute approximation DGS. \( \square \)

**Theorem 4:** For every constant \( k \) the \( k \)-absolute approximation DGS is strongly NP-hard even if each network contains only two types of gates, and each cell library contains only three cells with one cell to realize the gates of one type, and the other two to realize that of the other type. Furthermore, each cell area is limited to 1 and 2, and delay to 1 and 3.

**Proof:** Similar to that of Theorem 3, but with Construction 2 instead of Construction 1. \( \square \)

**IV. CONCLUSIONS**

We have shown the discrete gate-sizing problem is strongly NP-complete, and the \( k \)-absolute approximation discrete gate-sizing problem is strongly NP-hard for any \( k \). In the proof of both problems, the cell library under consideration is fixed and very small. In fact, it only contains a total of three cells. Each cell area is either 1 or 2, and each cell delay is either 1, 4 or 1, 3, depending on whether reconvergent paths are allowed in the network instance or not. It is not very clear that the discrete gate-sizing problem would remain to be NP-complete if each cell delay is further restricted to 1 and 2 or 1 and 3 for the case where no reconvergent path is allowed. Further study is needed to answer these open questions.

It is interesting to note that the network instance constructed using Construction 1 does not have any reconvergent path. It seems that the difficulty that makes the problem strongly hard comes from the set of gates that are on two or more of those paths that start from different primary inputs and terminate at different primary outputs. However, the occurrence of reconvergent paths may make the problem even harder, since with them each cell delay can be further restricted, and the problem remains strongly NP-hard. By focusing on how to deal with those gates and reconvergent paths, better heuristics tailored to this problem may be developed.

In our future research, we shall consider other kinds of approximation of the discrete gate-sizing problem. We shall also develop and study effective heuristics and approximation algorithms for this problem.

**ACKNOWLEDGMENT**

Thanks to Elizabeth Justice for preparing all the diagrams in this paper. The comments made by an anonymous referee on the delay models are appreciated.

**REFERENCES**


**An Observability Enhancement Approach for Improved Testability and At-Speed Test**

Elizabeth M. Rudnick, Vivek Chickereman, Member, IEEE, and Janak H. Patel, Fellow, IEEE

**Abstract—** Some recent studies show that an at-speed sequential or functional test is better than a test executed at lower speed. Design-for-testability approaches based on full scan, partial scan or silicon-based solutions such as Crosscheck achieve very high stuck-at fault coverage. However, in all these cases, the tests have to be applied at speeds lower than the operation speed of the circuit. In this paper, a design-for-test method that permits at-speed testing is introduced. The method is based on probe point insertion for improved observability, and it requires enhancements to an existing sequential circuit fault simulator. Faults that can be activated but not detected at existing primary outputs are targeted. A minimal set of probe points is selected to detect these faults, and the probe points are compressed to one or two output pins using exclusive-OR trees. The issue of aliasing of fault effects is addressed. Improvements in fault coverage were made for all 17 of the ISCAS89 sequential benchmark circuits studied. Fault coverages between 99% and 100% were obtained for seven circuits, and 98% ATG effectiveness was achieved on all but two circuits.

Manuscript received December 8, 1992; revised March 17, 1994. This work was supported by the Semiconductor Research Corporation under Contract SRC 92-10-109 and by a grant from Motorola. This paper was recommended by Associate Editor C. E. Kime.

E. M. Rudnick and J. H. Patel are with the Center for Reliable and High-Performance Computing, University of Illinois, Urbana, IL 61801 USA.

V. Chickereman was with the Center for Reliable and High-Performance Computing, University of Illinois, Urbana. He is now with IBM Corporation, Endicott, NY 13760 USA.

IEEE Log Number 9401866.