2016

Non-enumerative Generation of Path Delay Distributions and its Application to Critical Path Selection

Ahish Mysore Somashekhar
Southern Illinois University Carbondale, msahish@siu.edu

Spyros Tragoudas
Southern Illinois University Carbondale, spyros@siu.edu

Rathish Jayabharathi
Intel, rathish.jayabharathi@intel.com

Sreenivas Gangadhar
Intel, sreenivas.gangadhar@intel.com

Non-enumerative Generation of Path Delay Distributions and its Application to Critical Path Selection

Ahish Mysore Somashekar, Southern Illinois University
Spyros Tragoudas, Southern Illinois University
Rathish Jayabharathi, Intel Corporation
Sreenivas Gangadhar, Intel Corporation

A Monte Carlo based approach is proposed capable of identifying in a non-enumerative and scalable manner the distributions that describe the delay of every path in a combinational circuit. Furthermore, a scalable approach to select critical paths from a potentially exponential number of path candidates is presented. Paths and their delay distributions are stored in Zero Suppressed Binary Decision Diagrams. Experimental results on some of the largest ISCAS-89 and ITC-99 benchmarks shows that the proposed method is highly scalable and effective.

1. INTRODUCTION

As CMOS technology is scaled down in to the deep nanometer regime, control of the physical parameters, such as the feature size of transistors, their doping levels, and oxide thicknesses, has become extremely difficult. Also, the close proximity of devices to each other has given rise to significant interference from elements surrounding a device, due to inductive and capacitive coupling, and due to environmental factors such as power supply and temperature fluctuations. These and other factors have made it increasingly difficult to predict the timing behavior. This results in significant loss of yield of the manufactured circuits.

Due to process variations the delay of a path varies among the different manufactured chips. Therefore, the path delay is no more a discrete quantity but a statistical quantity, a.k.a a distribution. A path delay distribution can be obtained using SSTA [Blaauw et al. 2008] or Monte Carlo. Under the SSTA based technique, the delay of each gate/interconnect is given as a probability density function (pdf) represented using a mean (μ) and a standard deviation (σ). The delay of a path is derived by performing a statistical summation operation over all gates/interconnects along the path.

Monte Carlo, on the other hand, generates several instances of a given circuit. In each instance, a discrete value is assigned to each gate and interconnect of the circuit [Stein 1986]. The delay of a path in each instance is a discrete value which is the sum of all the discrete values corresponding to the gates and interconnects along the path. The delay distribution of a path is obtained by collecting its delays over all generated instances.

The above stated method for identifying the delay distribution of each path is path-enumerative in nature, and it is non-scalable given the potentially exponential number of paths in the circuit. Analyzing the delay distributions of paths is key to determining

This research has been supported in part by grants NSF IIP 1432026, and NSF IIP 1361847 from the NSF I/UCRC for Embedded Systems at SIUC. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation

DOI: http://dx.doi.org/10.1145/0000000.0000000

the operating frequency (clock period) and the consequent timing yield. This requires a framework to store and manipulate path delay distributions without path enumeration. Less pessimistic timing analysis methods as in [Sivaraman and Strojwas 2000] rely on Monte Carlo, and for each instance they operate on a path enumerative manner. Thus, they are not scalable. The proposed method may accelerate such efforts. After all pdfs are calculated with our method, one can identify the highest delay and all associated path frequencies. This will indicate the probability of the highest circuit delay. Similar information can be extracted for the second highest delay, and so on. Thus, the circuit's critical timing behavior can be obtained without path enumeration.

The second contribution in this article is aimed at selecting a set of critical paths under any clock period in a scalable manner without path enumeration. These are the set of paths that exhibit a high probability of violating the desired clock period due to defects arising from manufacturing process variations or signal integrity issues. The path delay fault (PDF) model is considered to be very effective in detecting failures caused by undesirable variations in manufacturing processes, [Wang et al. 2009] [Padmanaban and Tragoudas 2005] among others. It is prohibitive to test all paths, and only a subset of paths that exceed a given test margin τ are tested. However, typically, the number of paths that exceed the test margin may be prohibitive, [Wang et al. 2009] [Padmanaban and Tragoudas 2005] among others, and this requires an approach which does not enumerate paths.

In order to account for variations in manufacturing processes and other environmental sources statistical timing models are used to model gates delays and the circuit delay. With the introduction of statistical timing models the criticality/critical probability of a path x ($\varphi_\tau(x)$) is defined as the probability with which x exceeds the given τ.

The shrinking time to market has tightly constrained the test budget for detection of delay defects and allows for only a small number of paths to be tested. The challenge, therefore, is to select the set of paths that provide the maximum delay defect coverage under a given test budget.

The authors in [Li et al. 1989] [Murakami et al. 2000] [Shao et al. 2003] [Sharma and Patel 2002] [Qiu and Walker 2003] [Lu et al. 2005] propose to select a longest testable path through each gate. These approaches are path enumerative and lack an appropriate modeling for process variations.

Path correlation is classified into two categories, one is topological correlation and the other is spatial correlation. Topological correlation refers to shared gates and interconnects and spatial correlation stems from high process correlation in closely placed path components. According to the spatial correlation model, gates placed very close on the layout are highly correlated and as the distance between any two gates increases their correlation decreases. The authors in [Agarwal et al. 2003] present a method to capture such spatial correlation between gates by dividing the circuit layout into grids.

The authors in [Christou et al. 2010] present a non-enumerative technique for measuring path correlations. A numerical value representing the topological correlation among a given set of paths is computed. However, it is unclear how this approach can take into account path criticality under process variations and spatial correlation of processes to yield a high quality path set for delay test.

Analytical path selection methods in [Zolotov et al. 2010] [Chung et al. 2012] [He et al. 2013] assume a Normal distribution of gate delays and the linear relation of gate delays to the process parameter variations. Also, they are imprecise in their statistical maximum calculation. Such inaccuracies or simplifications may lead to inaccuracies in computing path criticalities and selecting high quality path sets for effective delay testing.

Monte Carlo, on the other hand, makes no simplifying assumptions on the distribution of process parameters or the models for gate/circuit delays. The accuracy of the Monte Carlo method is independent of the set of process variations and holds direct dependence on the sample size. This is favorable because the accuracy can be increased with an increase in sample size. It is regarded as the golden standard for validating approximation based tools such as statistical static timing analysis(SSTA), and recent work in [Singhee et al. 2008] [Veetil et al. 2011] present enhanced sampling methods that deliver reduced sample sizes to achieve a desired accuracy.

The authors in [Wang et al. 2004] proposed a Monte Carlo approach for critical path selection while considering topological and spatial path correlations. [Wang et al. 2004] selects paths based on their conditional criticality. Conditional criticality of a path is the probability that a path is critical in the currently uncovered process space, which also translates to a given path's ability to detect defects that weren't already detected by the previously selected paths. This approach requires identifying enumeratively an initial set of candidate critical paths. The initial set of paths may be prohibitive on modern circuits with many close-to-critical paths. Each path is tested on every circuit instance to determine the set of instances detected by the path which are then removed from consideration for subsequent paths.

A Monte Carlo based approach is proposed capable of identifying in a non-enumerative and scalable manner the distributions that describe the delay of every path in a combinational circuit. Furthermore, a scalable approach to select critical paths from a potentially exponential number of path candidates is presented. Paths and their delay distributions are stored in Zero Suppressed Binary Decision Diagrams (ZBDDs). This article presents a holistic content that elaborates on the procedures used for non-enumerative generation of path delay distributions and non-enumerative correlation aware critical path selection. It expands on preliminary versions in [Somashekar et al. 2012] and [Somashekar et al. 2015], and introduces new procedures for implicit manipulation of paths and their related distributions.

The rest of this article is organized as follows. Section 2 presents preliminary information related to ZBDDs and the method of representing path distributions in ZBDDs. Section 3 presents the proposed approach for non-enumerative generation of path delay distributions for all paths in the given circuit. Section 4 presents the proposed correlation aware critical path selection method to return paths that cover the various process instances. Experimental results in Section 5 show the effectiveness of the proposed approach in comparison to existing approaches and Section 6 concludes.

2. PRELIMINARIES ON ZBDDS

A ZBDD is a canonical representation of Boolean expressions that facilitates implicit manipulation of a collection of sets. If the elements of each set are gates then each set represents a path.

The operation on ZBDDs correspond to the basic set manipulation operations and the ZBDDs provide built-in operators to carry out such operations. These operators have been shown to be very fast [Minato 1993]. Below we list the operators used in this article.

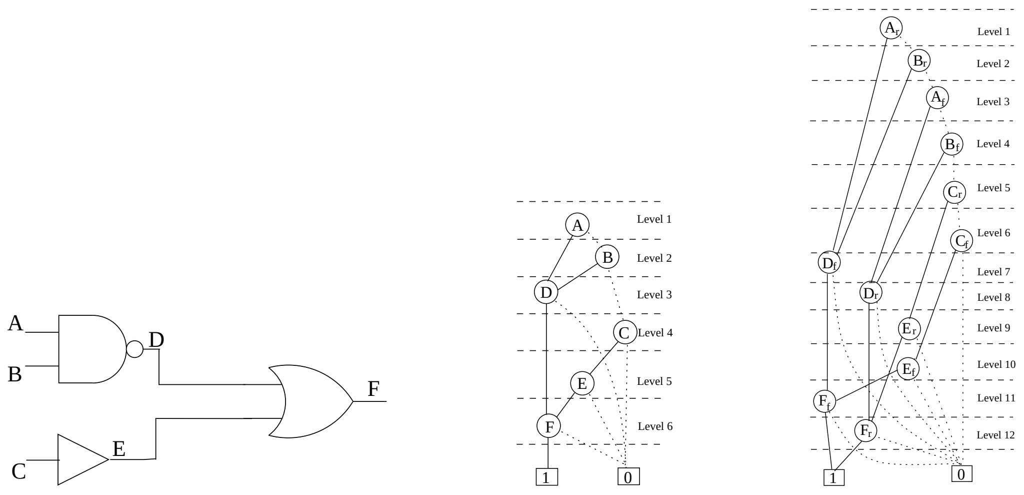
$$\begin{aligned}
\cup &\implies \text{Union}\\
\cap &\implies \text{Intersection}\\
*/\texttt{Update}() &\implies \text{Unate product}\\
- &\implies \text{Set difference}\\
\texttt{subset}() &\implies \text{Cofactoring}
\end{aligned}$$

The complexity of the procedures presented in this paper are expressed as a function of the number of calls to these operators.

Figure 1(b) shows a ZBDD containing all physical paths in the circuit shown in Figure 1(a). Each node in the ZBDD corresponds to a gate in the circuit. Nodes are placed in ZBDD levels, and all nodes at the same level correspond to the same gate or input pin in the circuit. In Figure 1(b), the root node is A, and it corresponds to input pin A. In the example of Figure 1(b), there are 3 inputs pins and 3 gates in the circuit, and the ZBDD has only 6 nodes that are placed in the 6 levels. However, in general, each ZBDD level may have multiple nodes that correspond to an input pin or gate. This will be shown in subsequent examples.

The solid edge out of each node indicates that the node belongs on a path. A dotted edge indicates that the node is not on a path. A ZBDD path is any directed path from the root to terminal $'1'$. In Figure 1(b), each ZBDD path corresponds to a physical path in the circuit. Given a circuit, the corresponding ZBDD is constructed using the operations given in [Padmanaban and Tragoudas 2005].

Figure 1(c) shows a ZBDD, denoted by Φ, representing the logical paths of the circuit in Figure 1(a). In Φ there are 2 nodes per physical circuit node. A subscript is used with node names to denote a transition at the respective nodes. The symbol r is used to denote a rising transition whereas the symbol f is used to denote a falling transition. Note that the ZBDD node corresponding to a rising transition at a gate and the ZBDD node corresponding to a falling transition at a gate are at different levels. This representation is useful to model different rise and fall delays.



(a) Circuit (b) Path ZBDD (c) Φ, Sensitizable paths

Fig. 1. ZBDDs of physical and sensitizable paths in a circuit

The delay distribution of a path represents the different delay values exhibited by the path and the number of times the path exhibited each delay value. A discerning approach to record delay distribution of a path is to observe the frequency f_k with which a path exhibits delays within each interval b_k. Figure 2 illustrates assuming a Normal distribution. However, the delay distribution of a path is not necessarily Normal and the proposed approach does not apply any approximation to normalize the distribution. The following describes a method to store path frequencies for different delay intervals in ZBDDs.

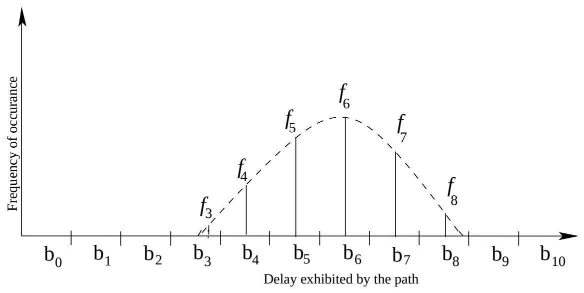


Fig. 2. Identifying the probability distribution of a path

We expand on the work in [Minato 1995] that presents an approach to represent polynomials in a ZBDD. A similar approach is employed to record the frequency of a path in a delay interval. To represent path delay distributions in a ZBDD additional nodes are added to Φ and paths are extended to include these nodes.

Consider the following expression

$$4ADF + 6BDF + 2CEF \tag{1}$$

The integer coefficients in the above expression can be represented as a sum of $2's$ exponents using binary encoding.

$$C = b_m \cdot 2^m + b_{m-1} \cdot 2^{m-1} + b_{m-2} \cdot 2^{m-2} .... + b_0 \cdot 2^0,$$

where $b \in \{0, 1\}$ and $\{b_m b_{m-1} b_{m-2} \ldots b_0\}$ is the m bit binary representation of the integer C. Integer 6 is represented in binary as 110. It is expressed using sum of $2's$ exponents as $\{2^2 + 2^1\}$. The expression in Equation 1 can be re-written as

$$2^2ADF + 2^1BDF + 2^2BDF + 2^1CEF$$

The ZBDD for the above expression is shown in Figure 3(a) where nodes '2^1' and '2^2' are added in order to associate frequency to paths. The frequency of any path is derived by adding the labels of all $2's$ exponent nodes with solid edges on the path.

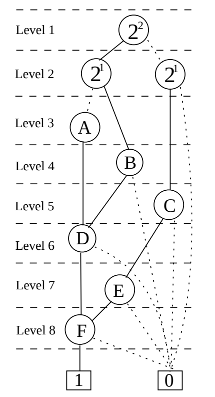
In general, nodes for the exponents of 2 are added to the ZBDD and are attached to the desired paths. These nodes are shared by all paths.

In the proposed approach, we first identify a set of delay intervals in the time line of the circuit delay. For k delay intervals we have k nodes in the ZBDD represented by b_i, i $\in \{0, 1, 2, \ldots k\}$.

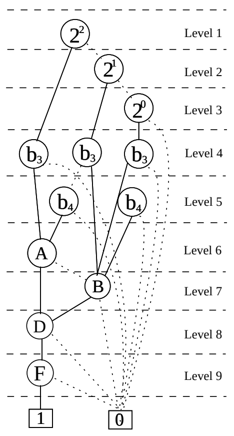
Nodes corresponding to delay intervals are added to the ZBDD and paths are tagged to these nodes to complete the representation of delay distributions. Consider path {A-D-F} with frequency 4 in interval b_3 and frequency 2 in b_4. Also, path {B-D-F} with frequency 3 in interval b_3 and 3 in interval b_4. The expression for these paths is

$$2^2b_3ADF + 2^1b_4ADF + 2^0b_3BDF + 2^1b_3BDF + 2^0b_4BDF + 2^1b_4BDF$$

Figure 3(b) shows the ZBDD that represents the above expression, where nodes 'b_3' and 'b_4' are added in order to associate delays with paths



(a) Polynomial paths in a ZBDD

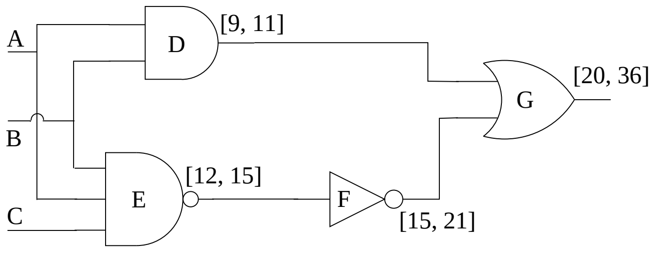


(b) Paths tagged with delay and frequency count

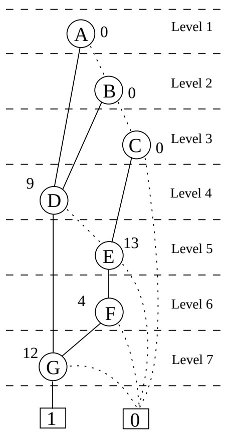
Fig. 3. Path specific information in ZBDD

3. PROPOSED APPROACH FOR NON-ENUMERATIVE GENERATION OF PATH DELAY DISTRIBUTIONS

This section presents an approach to identify delay distributions for an exponential number of paths in a circuit. This is achieved over several iterations. During each iteration the node/gate delays are instantiated using a Monte Carlo instance. At each iteration all path delay distributions are implicitly updated.



(a) Circuit showing arrival time bounds at output of each gate



(b) ZBDD representing a given circuit instance

Fig. 4. Implicit approach to identify path delays in a given circuit instance

Static timing analysis (STA) identifies the arrival (upper and lower) time bounds at the output of each node/gate, and, subsequently, a time interval on the circuit delay that represents the expected range of path delays. The time interval is represented with up to $\mathcal{P}$ discrete points that define the precision of the approach. $\mathcal{P}$ can be some exponent of 10. Assume a circuit operating at 1GHz frequency. The delay of the longest path in this circuit is at most $1ns$. If a precision of $\mathcal{P} = 10^3$ is used then two paths will be kept in distinct sets if their delay difference is at least $1ps$. In this scenario, there can be a maximum of 1000 discrete points to capture all path delays. Each discrete point, essentially, represents an interval that abstracts the delay to a given precision. The higher the precision the greater the accuracy with which we can capture path delay distributions. The choice of precision is a trade-off between accuracy and computation time.

A partial path is defined as a path from a primary input to an internal gate. We use ZBDDs to store partial paths. Partial paths are kept in different sets based on their delays. Discrete values are used to identify each set of partial paths.

Consider gate G shown in Figure 5. The ZBDD associated with the discrete point t at the output of G is represented as Ψ_G^t. Ψ_G^t will contain all partial paths up to G whose delay is t.

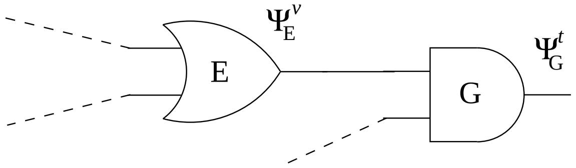


Fig. 5. Extending partial paths through a gate

Each gate in any given Monte Carlo instance has a fixed delay value. Let $\mathcal{I}(G)$ represent the delay of gate G in a given instance. In Figure 5, E is one of the inputs to G. Let Ψ_E^v be the ZBDD containing all those partial paths through E whose delay, in the given instance, ties to v. The following operations take place each time a partial path is extended through a gate G. Using Equation 2 the delay of the partial path, when extended to the output of G, is calculated. Then using Equation 3 the appropriate ZBDD is updated to include the partial path extended through G.

$$t = v + \mathcal{I}(G) \tag{2}$$

$$\Psi_G^t = \Psi_G^t \cup \left(\Psi_E^v * G\right) \tag{3}$$

`GenPathDelayDists()` in Procedure 1 lists the procedure to identify the path delay distributions in a given set of Monte Carlo instances. The inputs to Procedure 1 is a ZBDD Φ which implicitly holds all circuit paths, the desired precision $\mathcal{P}$ and the set of Monte Carlo instances $MC\text{-}Insts$.

The following describes Procedure 1 with an example. The following exposition uses the circuit in Figure 4(a) which shows the bounds on the arrival time at the output of each gate. Figure 4(b) shows the ZBDD containing paths from the circuit in Figure 4(a). The number adjacent to each node represents the corresponding gate delay in a given Monte Carlo instance.

Line 1 in Procedure 1 performs STA to determine the arrival time delay bounds at the output of each node. The function in line 2 identifies the discrete points within the delay range at each node based on the given precision. $\mathcal{A}$ is an associative array which stores the discreet points associated with each gate's output.

In lines $3-17$, the ZBDD(Φ) is traversed in a breadth first manner, i.e., level by level, from the root node to the terminal node 1 while processing nodes at each ZBDD level. In the ZBDD of Figure 4(b), A is the root node and the solid edge from A leads to node

Procedure 1: GenPathDelayDists

```
Input: $\Phi$, $\mathcal{P}$, $MC\text{-}Insts$
Output: $\xi$
1  $Delay\text{-}Bounds \leftarrow$ WC-STA($\Phi$)
2  $\mathcal{A} \leftarrow$ DiscretePoints($\Phi, Delay\text{-}Bounds, \mathcal{P}$)
3  foreach $inst \in MC\text{-}Insts$ do
4      $g \leftarrow$ Root($\Phi$)$.then$
5      while $g \neq Terminal\text{-}1$ do
6          foreach $i \in inedge(g)$ do
7              if $i = then$ then
8                  foreach $v \in \mathcal{A}(i)$ do
9                      $t \leftarrow v + \mathcal{I}(g)$
10                     $\Psi_g^t \leftarrow \Psi_g^t \cup$ Update($\Psi_i^v, g$)
11             else
12                 foreach $p \in inedge(i)$ do
13                     if $p = then$ then
14                         foreach $v \in \mathcal{A}(p)$ do
15                             $t \leftarrow v + \mathcal{I}(g)$
16                             $\Psi_g^t \leftarrow \Psi_g^t \cup$ Update($\Psi_p^v, g$)
17         $g \leftarrow g.next$
18     $\xi^{inst} \leftarrow \emptyset$
19     foreach $po \in PrimaryOutputs$ do
20         foreach $v \in \mathcal{A}(po)$ do
21             $b \leftarrow$ ZBDDvar ($v$)
22             $\xi^{inst} \leftarrow \xi^{inst} \cup$ Update($\Psi_{po}^v, b$)
23     $\xi \leftarrow$ UpdateDelayDists($\xi, \xi^{inst}$)
24 return $\xi$
```

D. For the given instance, the delay associated with node D is 9 and the parents of D are nodes A and B.

Nodes A and B are primary inputs and, in this scenario, both have delay 0. Therefore, all partial paths that extend to the output of node D tie to the same discrete point, i.e. 9. The only non-empty ZBDD at the output of D is expressed as follows

$$\Psi_D^9 = AD + BD$$

At the next level, node E has parents C and D. The delay associated with E in the given instance is 13 (see also Figure 4(b)). Node C has a solid (then) edge to E and therefore, it participates on the paths through E. The node D has a dotted edge (else) to E and therefore, it does not participate in the paths through E. We obtain the parents of node D with solid edge to D which are A and B.

Nodes A, B and C are primary inputs and, in this scenario, all three nodes have delay 0. Therefore, all partial paths that extend to the output of node E tie to the same discrete delay which in this case is 13. The only non-empty ZBDD at the output of E is expressed as

$$\Psi_E^{13} = AE + BE + CE$$

Node F has one parent node (E) and the delay at node F in the given instance is 4 (see also Figure 4(b)).

The delay of all partial paths through F is 17, and is obtained as follows

$$\mathcal{D}(\Psi_E^{13}) + \mathcal{D}(F) = 17$$

In the above expression, $\mathcal{D}(\Psi_E^{13})$ returns the corresponding discrete point 13. Ψ_E^{13} is the only non-empty ZBDD at the input of F and therefore, the only non-empty ZBDD at the output of F is expressed as follows

$$\Psi_F^{17} = AEF + BEF + CEF$$

Node G has solid edge parents D and F. The delay associated with G in the given instance is 12. The delay of all partial paths through input D is 21 at the output of G, which is obtained as follows

$$\mathcal{D}(\Psi_D^{9}) + \mathcal{D}(G) = 21$$

The delay of all partial paths through input F tie to 29 at the output of G, which is obtained as follows

$$\mathcal{D}(\Psi_F^{17}) + \mathcal{D}(G) = 29$$

The non-empty ZBDDs at the output of G are expressed as follows

$$\Psi_G^{21} = ADG + BDG$$
$$\Psi_G^{29} = AEFG + BEFG + CEFG$$

Let $\{b_0, b_1, \ldots b_{15}, b_{16}\}$ be the set of ZBDD variables that correspond to the discrete points that our method uses to denote discrete delays of paths from the circuit inputs to circuit outputs. In this example, we need 17 discrete points to represent all integers in the range [20-36] (see also output of gate G in Figure 4(a)). The final expression for the path delay distributions in the given instance is as follows

$$\xi^{inst} = b_1 ADG + b_1 BDG + b_9 AEFG + b_9 BEFG + b_9 CEFG$$

Line 23, in Procedure 1, calls `UpdateDelayDists()` to update ξ with the path delay information obtained from the current Monte Carlo instance. The complexity of Procedure 1, per Monte Carlo instance, is $O(E{\cdot}k{+}k{\cdot}m)$, in terms of built-in ZBDD operators, where E is the number of edges in the ZBDD, k is the number of discrete points to represent path delays and m is the number of frequency nodes. From this analysis one can observe that the proposed approach is capable of identifying the delay distributions of all paths in the circuit without enumerating the paths.

`UpdateDelayDists()` in Procedure 2 lists the sequence of operations carried out in order to update the path delay distributions after each Monte Carlo iteration.

The following illustrates `UpdateDelayDists()` (Procedure 2) with an example. Consider two paths {A-D-F, B-D-F}, the ZBDD in Figure 3 represents the delay distributions recoded for these paths up to the last Monte Carlo iteration and corresponds to ξ in the input to Procedure 2. Figure 6(a) shows the ZBDD of path delays obtained from the current Monte Carlo iteration and corresponds to ξ^{inst} in the input of Procedure 2. The ZBDD in Figure 6(a) suggests that for the given instance during the current Monte Carlo iteration paths A-D-F and B-D-F exhibited delays in intervals b_4 and b_5, respectively.

Procedure 2: UpdateDelayDists

```
Input: $\xi$, $\xi_{inst}$
Output: $\xi$
1  foreach $b \in \{b_0, \dots, b_n\}$ do
2      $\xi_b \leftarrow$ SubSet($\xi, b$)
3      $\xi^b_{inst} \leftarrow$ SubSet($\xi_{inst}, b$)
4      if $\xi^b_{inst} = \emptyset$ then
5          continue
6      $\xi^b_{inst} \leftarrow$ Update($\xi^b_{inst}, 2^0$)
7      if $\xi_b \neq \emptyset$ then
8          $\xi \leftarrow \xi - \xi_b$
9          $\xi_I \leftarrow \xi_b \cap \xi^b_{inst}$
10         $\xi_b \leftarrow (\xi_b \cup \xi^b_{inst}) - \xi_I$
11         $i \leftarrow 1$
12         while $\xi_I \neq \emptyset$ do
13             $\xi_{temp} \leftarrow$ SubSet ($\xi_I, 2^{i-1}$)
14             $\xi_{temp} \leftarrow$ Update($\xi_{temp}, 2^i$)
15             $\xi_I \leftarrow \xi_b \cap \xi_{temp}$
16             $\xi_b \leftarrow (\xi_b \cup \xi_{temp}) - \xi_I$
17             $i \leftarrow i + 1$
18     else
19         $\xi \leftarrow \xi \cup \xi^b_{inst}$
20     $\xi \leftarrow \xi \cup \xi_b$
21 return $\xi$
```

The expressions corresponding to the input ZBDD ξ is

$$\xi = 2^2 b_3 ADF + 2^1 b_4 ADF + 2^0 b_3 BDF + 2^1 b_3 BDF + 2^0 b_4 BDF + 2^1 b_4 BDF$$

and of ξ^{inst} is

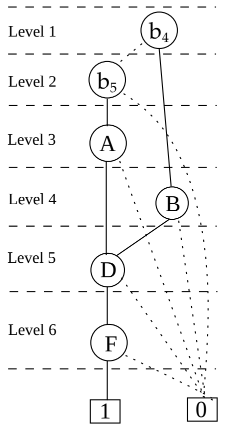
$$\xi^{inst} = b_4 BDF + b_5 ADF$$

For each interval b the procedure identifies the paths from ξ^{inst} and non-enumeratively updates the frequencies of the paths for the same interval in ξ. For the given instance no path exhibits delay in intervals b_0–b_3 and therefore, the process continues to b_4. In line 2 the function $\texttt{SubSet}(\xi, \texttt{b}_4)$ identifies the subset of paths in ξ that contain the variable b_4. The expression is

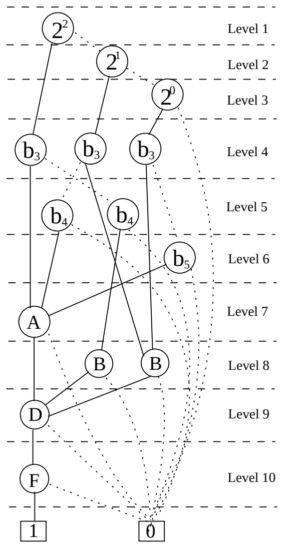
$$\xi_b = 2^1 b_4 ADF + 2^0 b_4 BDF + 2^1 b_4 BDF$$

Similarly, line 3 obtains the subset of paths with variable b_4 in ξ^{inst}. The expression is

$$\xi^{inst}_b = b_4 BDF$$



(a) ZBDD showing delay of paths in one Monte Carlo iteration



(b) ZBDD showing the updated path delay distributions from all Monte Carlo iterations

Fig. 6. Updating path distributions

Line 6 initializes a frequency count of 1 for all paths in $\xi_{b_4}^{inst}$ by implicitly augmenting each path with a variable corresponding to 2^0. This results in the following expression

$$\xi_b^{inst} = 2^0 b_4 BDF$$

Lines $8-17$ perform a series of operations to increment the frequency count of paths which is encoded using 2's exponents. In lines $8-10$ the paths that previously had a count of 1 (2^0) are identified. The following shows the corresponding expressions

$$\begin{aligned}\xi &= 2^2 b_3 ADF + 2^0 b_3 BDF + 2^1 b_3 BDF\\ \xi_I &= 2^0 b_4 BDF\\ \xi_b &= 2^1 b_4 ADF + 2^1 b_4 BDF\end{aligned}$$

The loop in lines $12-17$ increments the count of each path that previously exhibited delay in b_4 by 1. During the first iteration of the loop the variable corresponding to 2^0 is replaced with the variable corresponding to 2^1, effectively updating the path's frequency count from 1 to 2. The following shows the expressions corresponding to line $13-16$.

$$\begin{aligned}\xi_{temp} &= 2^0 b_4 BDF\\ \xi_{temp} &= 2^1 b_4 BDF\\ \xi_I &= 2^1 b_4 BDF\\ \xi_b &= 2^1 b_4 ADF\end{aligned}$$

The following shows the expressions corresponding to the lines $13-16$ during second iteration through the loop

$$\xi_{temp} = 2^1 b_4 BDF$$
$$\xi_{temp} = 2^2 b_4 BDF$$
$$\xi_I = \emptyset$$
$$\xi_b = 2^1 b_4 ADF + 2^2 b_4 BDF$$

Line 20 adds the updated interval b_4 back to the original ZBDD ξ.

$$\xi = 2^2 b_3 ADF + 2^1 b_4 ADF + 2^0 b_3 BDF + 2^1 b_3 BDF + 2^2 b_4 BDF$$

The process proceeds to interval b_5 and the following shows the expressions corresponding to the lines $2, 3$ and 6, respectively.

$$\xi_b = \emptyset$$
$$\xi_b^{inst} = b_5 ADF$$
$$\xi_b^{inst} = 2^0 b_5 ADF$$

The final updated ξ is expressed as follows

$$\xi = 2^2 b_3 ADF + 2^1 b_4 ADF + 2^0 b_5 ADF + 2^0 b_3 BDF + 2^1 b_3 BDF + 2^2 b_4 BDF$$

Figure 6(b) shows the pictorial representation of the final ξ.

The complexity of Procedure 2 is $O(k \cdot m)$, in terms of built-in ZBDD operators, where k is the number of intervals and m is the number of frequency nodes. Notice that the distributions of all paths are updated without enumerating the paths.

4. SELECTING CRITICAL PATHS FOR ATPG UNDER PATH CORRELATIONS

This section describes an approach to identify critical paths by implicitly considering path correlations. It is developed on the augmented data structure generated in Section 3. This augmented ZBDD is denoted by ξ.

The approach consists of three parts that are implemented without path enumeration. First, we consider all delay intervals that exceed test clock τ. They are merged into a single interval and the frequencies of the respective paths are updated. Then the highest frequency path is selected. Subsequent paths are selected by considering path correlations.

For a given test clock, the potential critical paths are those that exhibit delays greater than the test clock. The criticality or the critical probability of each path is given by the frequency with which the path exhibits a delay exceeding test clock. The frequency here is analogous to the number of defective instances that can be detected by testing this path. The critical probability for each path is implicitly held in the augmented ZBDD ξ.

First, procedure `MergeIntervals()` is presented. It is a method to combine paths in a given set of delay intervals and update their relative frequencies. This is a preprocessing step used for critical path selection. The input to the procedure `MergeIntervals()` is the ZBDD ξ which implicitly holds the paths delay distributions (Figure 3(b)), and the subset of intervals over which to combine paths and their frequencies. The output of the procedure is ZBDD ξ_U which is the union of all paths and their summed up frequencies over the given range of intervals. `MergeIntervals()` is outlined in Procedure 3 and is illustrated in the following with an example.

Consider the following expression corresponding to ZBDD in Figure 3(b)

$$2^2 b_3 ADF + 2^1 b_4 ADF + 2^0 b_3 BDF + 2^1 b_3 BDF + 2^0 b_4 BDF + 2^1 b_4 BDF$$

Procedure 3: MergeIntervals

```
Input: $\xi$, $\{b_i, \ldots, b_k\}$
Output: $\xi_U$
1  $\xi_U \leftarrow$ SubSet($\xi, b_i$)
2  foreach $b \in \{b_{(i+1)}, \ldots, b_k\}$ do
3      $\xi_b \leftarrow$ SubSet($\xi, b$)
4      $\xi_I \leftarrow \xi_U \cap \xi_b$
5      $\xi_U \leftarrow (\xi_U \cup \xi_b) - \xi_I$
6      while $\xi_I \neq \emptyset$ do
7          $\xi_{temp} \leftarrow$ UpdateCount($\xi_I$)
8          $\xi_I \leftarrow \xi_U \cap \xi_{temp}$
9          $\xi_U \leftarrow (\xi_U \cup \xi_{temp}) - \xi_I$
10 return $\xi_U$
```

The function `SubSet()` in line 1 co-factors ξ w.r.t b_3 and returns the subset of combinations/paths containing the variable/node b_3. It further performs an `UnateProduct` with b_U to yield

$$\xi_U = 2^2 b_U ADF + 2^0 b_U BDF + 2^1 b_U BDF$$

The above operations are carried out without path enumeration, and their time complexity is polynomial to the number of nodes in the ZBDD. Similarly, line 3 yields

$$\xi_b = 2^1 b_U ADF + 2^0 b_U BDF + 2^1 b_U BDF$$

The operations in line 4 and line 5 produce the following expressions

$$\begin{aligned} \xi_I &= 2^0 b_U BDF + 2^1 b_U BDF \\ \xi_U &= 2^2 b_U ADF + 2^1 b_U ADF \end{aligned}$$

Since ξ_I is not an empty set, the operation on line 7 `UpdateCount()` doubles the frequency count by replacing the nodes 2^i with 2^{i+1}. Notice that this is just a left shift of the binary representation, which is a multiply-by-2 operator. The output of lines 7, 8 and 9 are as follows

$$\begin{aligned} \xi_{temp} &= 2^1 b_U BDF + 2^2 b_U BDF \\ \xi_I &= \emptyset \\ \xi_U &= 2^2 b_U ADF + 2^1 b_U ADF + 2^1 b_U BDF + 2^2 b_U BDF \end{aligned}$$

Since ξ_I is now an empty set, the while loop is exited and the procedure in lines $3-9$ are repeated for other intervals in the list. In this case there are no more intervals and the combined entity ξ_U is obtained as shown above.

The complexity of Procedure 3 is $O(k \cdot m)$, in terms of built-in ZBDD operators, where k is the cardinality of the set of intervals given as input and m is the number of frequency nodes. Notice that the procedure is capable of updating the frequency of potentially an exponential number of paths across multiple intervals without path enumeration.

Using `MergeIntervals()`, the critical path selection operates only on one delay interval. In particular, all paths that exhibit delays greater than the test clock will be tagged to the node b_U and the combined frequency of each path in all intervals above the test clock will also be associated with the node b_U.

The remainder of this section describes procedure `ExtractCriticalPath`() that selects the path with the highest frequency in b_U. Such a path can potentially detect defects in most instances. Procedure 4 lists the process for extracting the highest frequency (most critical) path.

Procedure 4: ExtractCriticalPath

```
Input: $\xi_U$
Output: $\xi_\pi$
1 $\xi_\pi \leftarrow$ SubSet($\xi_U, 2^m$)
2 for $i \leftarrow m-1$ down to $0$ do
3     $\xi_{temp} \leftarrow$ SubSet($\xi_U, 2^i$)
4     if $(\xi_\pi \cap \xi_{temp}) \neq \emptyset$ then
5         $\xi_\pi \leftarrow \xi_\pi \cap \xi_{temp}$
6 return $\xi_\pi$
```

The input to procedure `ExtractCriticalPath`() is a ZBDD of paths with their associated frequencies in a single interval. The output is a set of one or more paths with the highest critical probability (frequency).

Line 1 initially obtains paths through the highest frequency node 2^m. This is a non-enumerative operation on the ZBDD which quickly returns a subset of paths whose frequency is at least 2^m. In lines $2-5$ the `SubSet`() operation is performed iteratively for all frequency nodes from 2^{m-1} down to 2^0. During each iteration, line 4 performs an intersection of paths through the current node with the previous set of paths. If this operation yields a non-empty set then line 5 updates the path set ξ_π to retain only the common set, which corresponds to the highest frequency paths up to the current node.

Let m be the number of frequency nodes, which is usually a very small number (much less than 20 for most circuits). The complexity of Procedure 4 is $O(m)$, in terms of built in ZBDD operators. Thus, the most critical path among a potentially exponential number of paths is identified without path enumeration.

This procedure is illustrated with an example below. Consider a set of paths { A-B-C, B-D-F, G-D-F, G-E-H } in the interval above the test clock. Consider the following expression which shows a certain frequency count associated with each path

$$2^0 b_U ABC + 2^5 b_U ABC + 2^6 b_U ABC + 2^1 b_U BDF + 2^5 b_U BDF + 2^6 b_U BDF + 2^4 b_U GDF + 2^3 b_U GEF$$

The approach starts from node 2^6 which yields paths {A-B-C, B-D-F}. This set of paths on intersection with the paths through node 2^5 would again yield {A-B-C,B-D-F}. For nodes 2^4 and 2^3, the intersection would yield $\emptyset$ (NULL) and the path set is unchanged. For node 2^1, the path set will be updated to be {B-D-F}. Again, intersection with paths for 2^0 returns a $\emptyset$. Thus, the path with the highest critical probability is {B-D-F}, and its frequency is $2^1 + 2^5 + 2^6$.

Subsequent paths are selected by considering path correlations. They should cover segments or process space not in the previously selected paths. All nodes in ξ_U are assigned a weight initialized to 1. During the path selection process the weight on nodes that physically participate, or have strong spatial correlation to other nodes, along previously selected paths is reduced by 1. After the weights are recomputed on each node the problem of new critical path selection reduces to the problem of selecting the longest path in a Directed Acyclic Graph (DAG).

In order to select paths with high critical probability, the longest path search is limited to paths above a user determined frequency threshold f_T. The procedure to quickly determine the set of paths with critical probability above f_T is listed in Procedure 5.

Procedure 5: ExtractFreqRange

```
Input: $\xi_U, f_T$
Output: $\xi_{f_T}$
1  $MSBpos \leftarrow$ GetMSB($f_T$)
2  $\xi_{temp} \leftarrow$ SubSet($\xi_U, 2^m$)
3  for $i \leftarrow m-1$ down to $MSBpos$ do
4      $\xi_{temp} \leftarrow \xi_{temp} \cup$ SubSet($\xi_U, 2^i$)
5  $\xi_{f_T} \leftarrow \xi_{temp}$
6  $\xi_{temp} \leftarrow$ SubSet($\xi_U, 2^{MSBpos}$)
7  $LSBpos \leftarrow$GetLSB($f_T$)
8  if $LSBpos \neq MSBpos$ then
9      for $i \leftarrow LSBpos$ to $MSBpos$-1 do
10         $\xi_{temp} \leftarrow \xi_{temp} -$ SubSet($\xi_U, 2^i$)
11     $\xi_{f_T} \leftarrow \xi_{f_T} - \xi_{temp}$
12 return $\xi_{f_T}$
```

The inputs to Procedure 5 are the augmented ZBDD ξ_U and the integer quantity f_T. The output is the set of all paths in ξ_U whose frequency is greater than f_T. The following illustrates the procedure with an example

Let m correspond to the highest frequency node present in ξ_U. In our example, this is node 2^6. Let f_T=10 and m=6. The 6-bit binary representation of 10 is 001010 which is represented as $2^3 + 2^1$.

The function `GetMSB(10)`, in line 1, identifies the highest 2's exponent for number 10, which is 3. The frequency of these paths is at least 2^3. In lines $2-5$ the procedure gathers all such paths and stores them in ξ_{f_T}.

In lines $6-11$, the procedure prunes out unsought paths with frequencies 8 and 9 which contain the frequency node 2^3.

The complexity of procedure `ExtractFreqRange`() is $O(m)$, in terms of built-in ZBDD operators. Its complexity is independent of the number of paths.

The approach presented above follows from the understanding that paths that are highly correlated share most of the same segments or contain segments that have strong spatial correlation and therefore, exhibit similar delay behavior. In other words, highly correlated paths have overlapping process space and tend to detect most of the same defective instances.

One may also insist that every new path should have at least $\mathcal{T}$ segments not in already selected paths, and f_T can be slowly relaxed until the desired path is selected. Identifying the optimal value of $\mathcal{T}$ can be challenging because it directly affects the correlation among the selected paths. Typically, $\mathcal{T}$ is set to a fraction of the average number of segments, denoted by α, along the critical paths in ξ_U. Quantity α is

$$\alpha = \frac{\sum\limits_{v \in V} \lambda_v}{|\xi_U|}, \tag{4}$$

where V is the set of all nodes in ξ_U and λ_v is the number of paths through v. Quantity λ_v, for each v, can be obtained in one forward and one backward traversal of ξ_U.

The proposed approach for correlation-aware path selection is listed in Procedure 6. The inputs to procedure `SelectPaths()` are the augmented ZBDD ξ (Section 3), the number of paths to select K, the set $\mathcal{D}_\tau$ of delay intervals above a given τ and the initial lower bound on the frequency range f_T. The output of `SelectPaths()` is a path set Π containing K paths.

Line 1, in `SelectPaths()`, calls `MergeIntervals()` (see Procedure 3). `MergeIntervals(`$\xi, \mathcal{D}_\tau$`)` returns a combined ZBDD of paths ξ_U and their associated frequencies over the given set of intervals $\mathcal{D}_\tau$. In line 2, `AvgPathLength(`ξ_U`)` returns the average number of gates/nodes along paths in ξ_U which is calculated using Equation 4. Line 3 calculates $\mathcal{T}$. w is a vector in which each element corresponds to the weight associated with a node in the ZBDD ξ_U. `InitNodeWeights()`, in line 4, initializes all elements in w to 1. In line 5, `ExtractCriticalPath()` (see Procedure 4) returns the path with the highest frequency. In line 6, the path p, from line 5, is added to Π. `UpdateNodeWeights()`, in line 7, reduces the weight corresponding to nodes that physically participate or have strong spatial correlation to nodes along the path p. `Eliminate(p)` in line 8 removes the path from ξ_U.

Lines $10 - 19$, in `SelectPaths()`, select the additional $K - 1$ paths. Line 10 calls `ExtractFreqRange(`ξ_U, f_T`)` (see Procedure 5) which returns a ZBDD of paths whose frequencies are greater than f_T. In line 11, `LongestPath()` is a linear time procedure that extracts the longest path in the ZBDD based on the weights in w. In line 12, `Length(p)` determines the length l of p as the sum of the weights on the nodes along p. If l is greater than $\mathcal{T}$ then p is added to Π else f_T is relaxed and the procedure is repeated until a total of K paths are selected.

Previous efforts, [Wang et al. 2004] [He et al. 2013], towards correlation aware path selection have relied on a small set of candidate paths because their approach for correlation aware path selection is path enumerative. These approaches are forced to select a small subset of candidate critical paths using heuristic pruning methods because they lacked the infrastructure to maintain the delay and critical probabilities for an exponential number of paths.

In fact, the approaches in [Wang et al. 2004] and [He et al. 2013] use Monte Carlo sampling and select only a restricted number of top delay paths from each sample. They combine the few paths selected from each sample and form their candidate set. These approaches do not explore all potential candidates and, as a result, the quality of the set of selected paths may suffer.

The following describes the metric used to demonstrate the effectiveness of the proposed path selection approach. The quality of a path set Π is denoted by $\mathcal{Q}(\Pi)$ and is evaluated based on the number of failing circuit instances the path set was able to detect among $10,000$ random circuit instances. A circuit instance is failing if the circuit delay exceeds the given τ. A path $p \in \Pi$ detects a failing instance if the path delay exceeds τ in the said failing instance. Let $\mathcal{N}$ denote the set of all failing instances and $\mathcal{N}(p)$ denote the set of failing instances detected by path p. The quality of a path set $\mathcal{Q}(\Pi)$ is calculated as follows

$$\mathcal{Q}(\Pi) = \frac{\left| \bigcup\limits_{\forall p \in \Pi} \mathcal{N}(p) \right|}{\left| \mathcal{N} \right|} \times 100 \tag{5}$$

Procedure 6: SelectPaths

```
Input: $\xi, K, \mathcal{D}_\tau, f_T$
Output: $\Pi$
1  $\xi_U \leftarrow$ MergeIntervals($\xi, \mathcal{D}_\tau$)
2  $\alpha \leftarrow$ AvgPathLength($\xi_U$)
3  $\mathcal{T} \leftarrow \lceil 0.5 \times \alpha \rceil$
4  $w \leftarrow$ InitNodeWeights()
5  $p \leftarrow$ ExtractCriticalPath($\xi_U$)
6  $\Pi \leftarrow p$
7  $w \leftarrow$ UpdateNodeWeights($p$)
8  $\xi_U \leftarrow$ Eliminate($p$)
9  $K \leftarrow K - 1$
10 while $K \neq 0$ do
11     $\xi_{f_T} \leftarrow$ ExtractFreqRange($\xi_U$, $f_T$)
12     $p \leftarrow$ LongestPath($\xi_{f_T}, w$)
13     $l \leftarrow$ Length($p$)
14     if $l > \mathcal{T}$ then
15         $\Pi \leftarrow \Pi \cup p$
16         $w \leftarrow$ UpdateNodeWeights($p$)
17         $\xi_U \leftarrow$ Eliminate($p$)
18         $K \leftarrow K - 1$
19     else
20         $f_T \leftarrow$ Relax()
21 return $\Pi$
```

5. EXPERIMENTAL RESULTS

The experimental results highlight the scalability of the proposed approach in terms of generating the path delay distributions for all paths in the circuit and subsequently, selecting a set of critical paths for testing from among a potentially exponential number of candidate paths.

Experimental evaluation was performed on circuits from the ISCAS' 85, ISCAS' 89 and ITC' 99 benchmarks. Experiments were conducted on a Linux machine with Intel Xeon processor and 24 Gb memory. The presented procedures were implemented in the 'C++' programming language. The nominal delay for each gate was obtained from a 45nm technology library. To account for spatial correlations, the circuit was partitioned into several grids and gates were randomly assigned to each grid. We generated $10,000$ Monte Carlo instances by assuming 5% variation in oxide thickness T_{ox} and 10% variation in gate length L. Gates assigned to the same grid received identical process shifts. Table I lists the time for generating the delay distributions of all paths with $1ps$ delay precision.

Figure 7 illustrates the trade-off between time complexity of the approach and the granularity of path delay distributions. As shown in Figure 7, the complexity of the proposed approach increases almost linearly with increasing precision (decreasing granularity) of desired path delay distributions. Observe that the time penalty for the proposed approach is very reasonable considering the number of paths in the corresponding circuits (see column 3 of Table I).

The test clock (threshold τ) was set to 90% of the worst case circuit delay. The worst case circuit delay was calculated using static timing analysis. The quantity f_T which

Table I. Implicit generation of Path delay distributions with $1ps$ granularity

Benchmark	Nodes	Paths	Time
C5315	4970	2.68E+06	2674.36
C6288	4896	1.97E+20	3147.82
C7552	7440	1.45E+06	3100.02
S38417	51394	2.78E+06	6740.6
S38584	44646	2.16E+06	5071.2
b05	2064	3.99E+08	2042.83
b15	18742	9.65E+10	11968.12
b17	67482	7.91E+11	29884.9
b18	242482	2.26E+24	128369.3
b19	489148	1.09E+25	212366.1
b20	41432	5.79E+08	14113.2
b21	42122	5.56E+08	15629.8
b22	61372	8.74E+08	22184.3

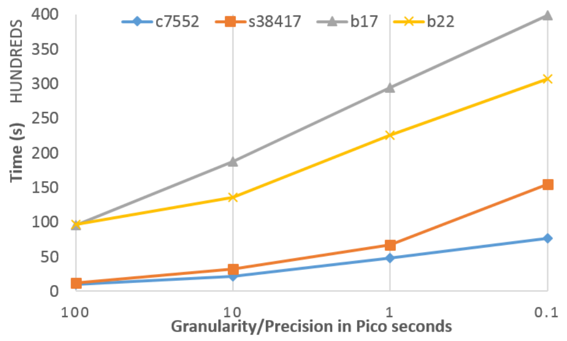


Fig. 7. Trade-off between time and accuracy

Table II. Selecting 10 paths under different values for quantity $\mathcal{T}$

Benchmark	Grids	Total Paths	Critical Paths	$\mathcal{Q}(\Pi)$		
				$\mathcal{T} = \lceil 0.3 \times \alpha \rceil$	$\mathcal{T} = \lceil 0.5 \times \alpha \rceil$	$\mathcal{T} = \lceil 0.8 \times \alpha \rceil$
C5315	4×4	2.68E+06	759680	72.69	100	98.5
C6288	4×4	1.98E+20	1.08E+19	57.13	88.34	41.12
C7552	8×8	1.45E+06	222732	67.52	99.01	98.12
S38417	16×16	2.78E+06	408759	81.47	95.87	82.13
S38584	16×16	2.16E+06	223210	79.09	96.33	88.7
b05	8×8	3.99E+08	3.76E+06	91.54	99.7	94.25
b15	12×12	9.65E+10	1.78E+10	87.45	99.16	72.63
b17	20×20	7.91E+11	2.90E+10	75.48	95.92	82.12
b18	40×40	2.26E+24	1.09E+21	45.52	88.19	66.23
b19	60×60	1.09E+25	9.13E+22	51.08	83.62	57.17
b20	20×20	5.79E+08	4.47E+07	65.28	96.6	57.64
b21	20×20	5.56E+08	4.48E+07	63.48	97.36	44.12
b22	20×20	8.74E+08	6.69E+07	78.87	96.05	55.28

determines the lower bound on frequency was relaxed by 100 units if a suitable path was not found in the current iteration.

The proposed approach for path selection (see Procedure 6) was compared against the Monte Carlo (MC10000) based explicit path selection approach using $10,000$ instances, as in [Wang et al. 2004]. We call this method MC10000, and operates as follows. First, a candidate set U is generated by collecting all paths that exhibit delays above the given τ. Paths in U are ordered based on their probability of exceeding τ. A path p from U is selected, starting with the path that has the highest probability, and is tested over all instances. Paths are then ranked based on the number of samples detected by each path. Circuit samples detected by path p are deleted before considering the next path.

Note that, if critical path selection is the only objective and the test clock τ is perdetermined then it is not necessary to generate the delay distribution of paths over all intervals. The authors in [Padmanaban and Tragoudas 2005] provide an efficient non-enumerative approach to quickly identify all paths that exceed a given τ in a given Monte Carlo sample. The approach in [Padmanaban and Tragoudas 2005] is used along with `UpdateDelayDists()` (see Procedure 2) to generate the critical frequencies (critical probabilities) of the paths. This way we consider a single interval above τ and in this context, the procedure `MergeIntervals()` (see Procedure 3) is not required. The critical path selection algorithm has been implemented using this approach.

Our experimental analysis for the proposed critical path selection approach (Procedure 6) is presented in the following three tables. Table II presents the quality of a set of 10 paths selected under different values of quantity $\mathcal{T}$. Table III and Table IV present results for path selection using the proposed approach in Procedure 6 when compared to path selection using the path enumerative Monte Carlo based approach MC10000.

In Table II, Column 1 lists the circuit name. Column 2 lists the number of grid partitions per circuit in order to account for spatial correlations. Column 3 lists the total number of paths in the circuit. Column 4 lists the number of paths that exhibited delays greater than τ i.e., at least 90% of the worst case circuit delay. Columns $5-7$ list the quality factor, given by Equation 5, under different $\mathcal{T}$ (described in Section 4). We choose three different values for $\mathcal{T}$ and list the quality factor of the selected path set under each. One can observe that the approach performs best when $\mathcal{T} = \lceil 0.5 \times \alpha \rceil$. A lower value on $\mathcal{T}$, as in column 4, requires only a few uncovered segments along every new path. The paths selected, in column 4, were highly correlated with fewer additional instances detected by each new path. A higher value on $\mathcal{T}$, as in column 6, demands a larger number of uncovered segments along every new path. In most cases, the approach was unable to find paths that satisfied this criterion and therefore, the final tally of paths, after several reductions of f_T, was still under 10. In fact, the approach returned 10 paths only for C7552.

Table III. Comparative results for selecting 10 paths from the set of all critical paths

Benchmark	Critical Paths	`SelectPaths()`		MC10000, as in [Wang et al. 2004]	
		$\mathcal{Q}(\Pi)$	Time (s)	$\mathcal{Q}(\Pi)$	Time (s)
C5315	759680	100	170.98	100	114076
C6288	1.08E+19	88.34	3102.11	-	-
C7552	222732	99.01	435.4	99.82	45351
S38417	408759	95.87	1439.78	96.68	92580
S38584	223210	96.33	1313.42	95.64	42996
b05	3.76E+06	99.7	114.56	-	-
b15	1.78E+10	99.16	1648	-	-
b17	2.90E+10	95.92	4205.07	-	-
b18	2.26E+24	88.19	14496.18	-	-
b19	1.09E+25	83.62	18037.20	-	-
b20	4.47E+07	96.6	4454.06	-	-
b21	4.48E+07	97.36	4303.18	-	-
b22	6.69E+07	96.05	9257.3	-	-

In Table III, column 1 lists the circuit name. Column 2 lists the the number of paths that exhibited delays greater than τ i.e, at least 90% of the worst case circuit delay. Column 3 and 4 list the quality factor, with $\mathcal{T} = \lceil 0.5 \times \alpha \rceil$, and time for the proposed approach to select a set of 10 testable paths from the set of all critical paths in the circuit. Note that the time reported here is cumulative of the time taken for generating the critical frequencies of all potentially critical paths considering 10000 samples and the test generation for each selected path. The testability of a path is determined after the path is selected using [Padmanaban and Tragoudas 2005], [Kim et al. 2000]. The path selection is carried out over several iterations. During each iteration one path is selected using the proposed approach and untestable paths were discarded. This procedure of determining the testability of a path applies to MC10000 as well. The quality factor and time for MC10000 are listed in columns 5 and 6, respectively. For the ISCAS circuits, the quality of paths selected by the proposed approach is the same as that of MC10000 and the proposed approach achieves an average speed up of $100\times$ when compared to MC10000. One can observe that as the number of candidate paths increases, as is the case for ITC'99 benchmarks, the approach in MC10000 becomes prohibitive.

In order to provide a more thorough comparative analysis on the quality of the path set selected by the proposed approach, we reduce the set of candidate paths by considering only the testable critical paths as in [Padmanaban and Tragoudas 2005]. Results from this analysis are listed in Table IV.

In Table IV, column 1 lists the circuit name. Column 2 lists the number of testable critical paths. Columns 3 and 4 list the quality factor, with $\mathcal{T} = \lceil 0.5 \times \alpha \rceil$, and time for the proposed approach to select a set of 10 paths. Note that the time reported here is cumulative of the time taken for generating the critical frequencies of all potentially critical paths considering 10000 samples and the subsequent path selection process. Columns 5 and 6 list the quality factor and time for MC10000. When compared to the enumerative Monte Carlo approach for path selection, in columns 5 and 6, one can observe that the proposed approach achieves the same quality factor and the proposed approach is on an average $100\times$ faster than MC10000.

Table IV. Comparative results for selecting 10 paths from testable critical paths as in [Padmanaban and Tragoudas 2005]

Benchmark	Testable critical paths	`SelectPaths()`		MC10000, as in [Wang et al. 2004]	
		$\mathcal{Q}(\Pi)$	Time (s)	$\mathcal{Q}(\Pi)$	Time (s)
C5315	58110	99.36	137.92	100	3800
C6288	1.4E+07	91.32	317.38	-	-
C7552	86998	98.02	201.32	99.58	5236
S38417	185485	94.8	1218.58	95.56	32515
S38584	150148	93.33	1139.54	95.13	26686
b05	1010928	98.27	82.08	98.77	166932
b15	5242925	98.76	1588.93	98.91	666091
b17	1923882	96.14	3857.22	97.27	262589
b18	5.41E+11	87.51	8366.77	-	-
b19	3.96E+09	82.89	12042.91	-	-
b20	1857401	96.59	3562.36	96.42	240472
b21	7533892	97.23	3506.9	98.2	706840
b22	3790663	96.43	7192.5	96.5	408414

6. CONCLUSION

In this paper, a non-enumerative approach to identify the delay distributions of every path in the circuit was provided. Furthermore, a novel scalable approach towards path selection for effective delay testing is presented. It explores an exponential number of candidate paths but its time complexity is linear to the number of selected paths. It has been experimentally verified that the quality of the approach is similar to path enumerative methods.

REFERENCES

Aseem Agarwal, David Blaauw, and Vladimir Zolotov. 2003. Statistical Timing Analysis for Intra-Die Process Variations with Spatial Correlations. In *Proceedings of the 2003 IEEE/ACM International Conference on Computer-aided Design (ICCAD '03)*.

D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer. 2008. Statistical Timing Analysis: From Basic Principles to State of the Art. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 27, 4 (April 2008), 589–607. DOI:http://dx.doi.org/10.1109/TCAD.2007.907047

K. Christou, M.K. Michael, and S. Neophytou. 2010. Identification of critical primitive path delay faults without any path enumeration. In *VLSI Test Symposium (VTS), 2010 28th*. 9–14. DOI:http://dx.doi.org/10.1109/VTS.2010.5469629

Jaeyong Chung, Jinjun Xiong, V. Zolotov, and J.A. Abraham. 2012. Testability-Driven Statistical Path Selection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 8 (Aug 2012), 1275–1287.

Zijian He, Tao Lv, Huawei Li, and Xiaowei Li. 2013. Test Path Selection for Capturing Delay Failures Under Statistical Timing Model. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 21, 7 (July 2013), 1210–1219. DOI:http://dx.doi.org/10.1109/TVLSI.2012.2208661

Joonyoung Kim, Jesse Whittemore, Karem A. Sakallah, and João P. Marques Silva. 2000. On Applying Incremental Satisfiability to Delay Fault Testing. In *2000 Design, Automation and Test in Europe (DATE 2000), 27-30 March 2000, Paris, France*. 380–384. DOI:http://dx.doi.org/10.1109/DATE.2000.840299

Wing-Ning Li, S.M. Reddy, and S.K. Sahni. 1989. On path selection in combinational logic circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 8, 1 (Jan 1989), 56–63. DOI:http://dx.doi.org/10.1109/43.21819

Xiang Lu, Zhuo Li, Wangqi Qiu, D.M.H. Walker, and Weiping Shi. 2005. Longest-path selection for delay test under process variation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 24, 12 (Dec 2005), 1924–1929. DOI:http://dx.doi.org/10.1109/TCAD.2005.852674

S. Minato. 1993. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In *Design Automation, 1993. 30th Conference on*. 272–277. DOI:http://dx.doi.org/10.1109/DAC.1993.203958

S. Minato. 1995. Implicit manipulation of polynomials using zero-suppressed BDDs. In *European Design and Test Conference, 1995. EDTC 1995, Proceedings.* 449–454. DOI:http://dx.doi.org/10.1109/EDTC.1995.470321

Atsushi Murakami, Seiji Kajihara, Tsutomu Sasao, Irith Pomeranz, and Sudhakar M. Reddy. 2000. Selection of potentially testable path delay faults for test generation. In *Proceedings IEEE International Test Conference*. 376–384.

S. Padmanaban and S. Tragoudas. 2005. Efficient identification of (critical) testable path delay faults using decision diagrams. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 24, 1 (Jan 2005), 77–87. DOI:http://dx.doi.org/10.1109/TCAD.2004.839488

Wangqi Qiu and D. M. H. Walker. 2003. An Efficient Algorithm for Finding the K Longest Testable Paths Through Each Gate in a Combinational Circuit. In *Proceedings of IEEE International Test Conference*. 592–601.

Yun Shao, Sudhakar M. Reddy, Irith Pomeranz, and Seiji Kajihara. 2003. On Selecting Testable Paths in Scan Designs. *Journal of Electronic Testing* 19, 4 (2003), 447–456.

M. Sharma and J.H. Patel. 2002. Finding a small set of longest testable paths that cover every gate. In *Test Conference, 2002. Proceedings. International*. 974–982. DOI:http://dx.doi.org/10.1109/TEST.2002.1041853

Amith Singhee, Sonia Singhal, and Rob A. Rutenbar. 2008. Practical, fast Monte Carlo statistical static timing analysis: Why and how. *Computer-Aided Design, International Conference on* 0 (2008), 190–195. DOI:http://dx.doi.org/10.1109/ICCAD.2008.4681573

M. Sivaraman and A.J. Strojwas. 2000. Primitive path delay faults: identification and their use in timing analysis. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 19, 11 (Nov 2000), 1347–1362. DOI:http://dx.doi.org/10.1109/43.892858

Ahish Mysore Somashekar, Spyros Tragoudas, Sreenivas Gangadhar, and Rathish Jayabharathi. 2012. Non-enumerative generation of statistical path delays for ATPG.. In *Proceedings IEEE International Con-*

ference on Computer Design (ICCD). IEEE Computer Society, 514–515. http://dblp.uni-trier.de/db/conf/iccd/iccd2012.html#SomashekarTGJ12

Ahish Mysore Somashekar, Spyros Tragoudas, and Rathish Jayabharathi. 2015. Non-Enumerative Correlation-Aware Path Selection. In *Proceedings IEEE International Conference on Computer Design (ICCD)*. IEEE Computer Society.

M.L. Stein. 1986. An Efficient Method of Sampling for Statistical Circuit Design. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 5, 1 (January 1986), 23–29. DOI: http://dx.doi.org/10.1109/TCAD.1986.1270174

V. Veetil, K. Chopra, D. Blaauw, and D. Sylvester. 2011. Fast Statistical Static Timing Analysis Using Smart Monte Carlo Techniques. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 30, 6 (June 2011), 852–865. DOI: http://dx.doi.org/10.1109/TCAD.2011.2108030

Li-C. Wang, Jing-Jia Liou, and Kwang-Ting Cheng. 2004. Critical path selection for delay fault testing based upon a statistical timing model. *IEEE Trans. on CAD of Integrated Circuits and Systems* 23, 11 (2004), 1550–1565. http://dblp.uni-trier.de/db/journals/tcad/tcad23.html#WangLC04

Laung-Terng Wang, Yao-Wen Chang, and Kwang-Ting (Tim) Cheng. 2009. *Electronic Design Automation, Elsevier, First Edition, 2009*.

V. Zolotov, Jinjun Xiong, H. Fatemi, and C. Visweswariah. 2010. Statistical Path Selection for At-Speed Test. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29, 5 (May 2010), 749–759.