Fast and Accurate Statistical Criticality Computation under Process Variations

Hushrav D Mogal, Haifeng Qian, *Member, IEEE,* Sachin S Sapatnekar, *Fellow, IEEE,* and Kia Bazargan, *Member, IEEE*

***Abstract*— With ever shrinking device geometries, process variations play an increased role in determining the delay of a digital circuit. Under such variations, a gate may lie on the critical path of a manufactured die with a certain probability, called the criticality probability. In this paper, we present a new technique to compute the statistical criticality information in a digital circuit under process variations by linearly traversing the edges in its timing graph and dividing it into "zones". We investigate the sources of error in using tightness probabilities for criticality computation with Clark's statistical maximum formulation. The errors are dealt with using a new clustering based pruning algorithm which greatly reduces the size of circuit-level cutsets improving both accuracy and runtime over the current state of the art. On large benchmark circuits, our clustering algorithm gives about a $250X$ speedup compared to a pairwise pruning strategy with similar accuracy in results. Coupled with a localized sampling technique, errors are reduced to around 5% of Monte Carlo simulations with large speedups in runtime.**

I. Introduction and Previous Work

With scaling technology trends, process parameter variations render the delay of the circuit as unpredictable [10], making sign-off ineffective in assuring against chip failure. Moreover, conventional Static Timing Analysis (STA) is unable to cope with a large process corner space. To tackle this problem, over the recent years, CAD tools have accounted for variability in the design flow. Of foremost concern is to predict the circuit delay in the face of these process parameter variations. Recent works concerning Statistical Static Timing Analysis (SSTA) in [1], [13] deal with this issue by treating the delay of gates and interconnects as random variables with Gaussian distributions. The techniques developed effectively predict the mean and variance of circuit delay distribution to an accuracy level of under a few percent.

The unpredictability in circuit delay also undermines design optimizations with timing considerations. For one, a gate sizing operation typically proceeds by finding the most critical path in a circuit and sizing the gates on this critical path. With process variations in a design, no one path dominates the delay of the circuit [8]. We therefore need the notion of probability to make informed decisions as to the relative importance of different gates in a design. The authors in [13] propose the concept of a path criticality, which is the probability that a path in the manufactured chip is critical. This concept is also extended to edge (node) criticalities in the timing graph of a circuit, i.e., the probability that a path passing through the edge (node) is critical. To this end, works like [2], [8] and [15] attempt to compute the criticality probability of edges in a timing graph, using a canonical first order delay model.

One of the earliest attempts to compute edge criticalities was proposed in [13]. The authors in [13] perform a reverse traversal of the timing graph, multiplying criticality probabilities of nodes with local criticalities of edges, incorrectly assuming the independence of edge criticalities despite structural and spatial correlations in the circuit. Subsequently, the work in [8] defined the statistical sensitivity matrix of an edge in the timing graph with respect to the circuit output delay, computed by using the chain rule in the backward propagation of the timing graph. Due to the matrix multiplications involved, the complexity of their approach, although linear in circuit size, could be potentially cubic in the number of principal components if the matrices are not sparse.

In [2] the authors perturb gate delays to compute its effect on the circuit output delay. The complexity of computation is reduced using the notion of a cutset belonging to a node in the timing graph. A cutset is a minimal set of edges, the removal of which divides the timing graph into two disconnected components. A key property is that the statistical maximum of the sum of arrival and required times across all the edges of a cutset gives the circuit delay distribuition. A gate sizing operation on the source node of a cutset affects only the arrival time of some of its edges. The circuit delay is then incrementally updated to efficiently compute the circuit yield gradient to gate sizing. This approach however, is potentially quadratic in the size of the timing graph.

The cutset-based idea is extended in [15], to compute the criticality of edges by linearly traversing the timing graph. The criticality of an edge in a cutset is computed using a balanced binary partition tree. Edges recurring in multiple cutsets are kept track of in an array based structure while performing the timing graph traversal.

This paper, a preliminary version of which appears in [9], makes the following contributions. First, similar to [15], we propose an algorithm to compute the criticality probability of edges (nodes) in a timing graph using the notion of cutsets. Edges crossing multiple cutsets are dealt with using a zone-based approach, similar to [16], in which old computations are reused to the greatest possible extent. Second, unlike [9] we investigate the effect of independent random variations on criticality computation and devise a simple scheme to keep

H. Mogal, S. Sapatnekar and K. Bazargan are with the Electrical Engineering Department, University of Minnesota, Minneapolis, MN 55455.

H. Qian is with IBM Research, Yorktown Heights, NY.

This work was supported in part by the SRC under award 2007-TJ-1572.

track of structural correlations due to such variations. Third, and more importantly, we examine the sources of error in criticality computations due to Clark's [3] formulation and propose a clustering based pruning algorithm to effectively eliminate a large number of non-competing edges in cutsets with several thousand edges. The proposed scheme can also help order statistical maximum operations in a set, a source of significant error as shown in [12]. Localized sampling on the pruned cutset further reduces errors in edge criticalities to within 5% of Monte Carlo simulations, with large speedups in runtime compared to a pairwise pruning strategy. Finally, we compare the clustering scheme with our implementation of [15] to show the improvement in runtime and accuracy.

The rest of this paper is organized as follows. Section II describes the correlation model and provides definitions used in the paper. Section III details our approach to compute edge criticalities using zones. Section IV discusses the errors involved in criticality computation. Our clustering based framework for criticality computation is described in Section V. Section VI gives details about techniques used by our algorithm to further reduce errors. We present the results in Section VII, followed by the conclusion in Section VIII.

II. Background

This section briefly describes the correlation model used to capture the process variations and provides definitions which are used throughout the rest of the paper.

A. Correlation Model and SSTA

We use the spatial correlation model in [1] to model intra-die parameter correlations. Briefly, the chip is divided into a uniform grid in which gates in a grid square are assumed to have perfect correlation and gates in far-away grid squares have weak correlations. A covariance matrix of size equal to the number of grid squares is obtained for each modeled parameter. The model includes a global (inter-die) component of variation common to each entry of the matrix. We model the length and width of gates, and the thickness, width and inter-layer dielectric thickness for interconnects. It is assumed that zero-correlations exist between different types of parameters, for example the length and width of a gate. We also model the gate oxide thickness, t_{ox}, assuming independence between the different gates on the chip. Although we do not model random dopant fluctuations in this work, they can be dealt with in a manner similar to t_{ox}. All process parameters are assumed to have a Gaussian distribution.

Delays of the edges in the timing graph (from gate fanins to gate outputs and from gate outputs to gate fanouts) are modeled in terms of their first order Taylor series expansions with respect to the process parameters. The correlated process parameters are orthogonalized using the principal component analysis (PCA) technique wherein each correlated parameter is expressed as a sum of independent and identically distributed normal random variables, called the principal components (PCs). Substituting the PCs back into the Taylor expansion gives a first order canonical edge delay model. SSTA is then performed by a forward propagation on the timing graph, as in a regular STA. For more details, readers are referred to [1].

B. Definitions

Definition II.1 (**Timing Graph**)**.** A timing graph $G(V, E)$ of a circuit is a directed acyclic graph with V nodes representing gate terminals and E edges representing connections between the terminals. Primary inputs and outputs are connected, respectively, to a virtual source node, v_s, and a virtual sink node, v_t. The delay of an edge in G is a random variable with an associated probability density function (pdf).

Definition II.2 (**Cutset**)**.** A cutset, Σ, is a set of edges/nodes in G such that **every** v_s to v_t path passes exclusively through a single member of Σ.

Definition II.3 (**Arrival Time (Required Time)**)**.** The arrival time AT_i^σ (required time RT_i^σ) at an edge/node, e_i, in timing graph, G, is the statistical maximum delay from any primary input (output) to the edge/node. Like delays, these are also random variables and have associated $pdfs$. Traditionally, $RAT_i^\sigma = T - RT_i^\sigma$, where RAT_i^σ is the required arrival time at e_i and T is the circuit timing specification.

Definition II.4 (**Path Delay**)**.** The path delay of an edge/node e_i in G, denoted e_i^σ, is defined as the sum of its arrival and required times and is a random variable with a pdf.

$$e_i^\sigma = AT_i^\sigma + RT_i^\sigma \tag{1}$$

In other words, the path delay of an edge/node represents the statistical maximum delay of all paths passing through it. Each path delay e_i^σ is represented in canonical form in terms of the independent and identical principal components (PCs) p_j, as

$$e_i^\sigma \quad = \quad \mu_i + \textstyle\sum_{j=1}^{j=k} a_{ij} \cdot p_j + \zeta_i \cdot r_i \tag{2}$$

Here a_{ij} is the sensitivity of edge e_i to PC p_j and k is the total number of PCs. The mean of the edge path delay is given by μ_i. Every edge e_i is also associated with a random uncorrelated component r_i with sensitivity ζ_i.

Definition II.5 (**Complementary Path Delay**)**.** Given a cutset, Σ, in a timing graph, G, the complementary path delay, $e_{i'}^\sigma$, of an edge/node $e_i \in \Sigma \subset G$, is defined as

$$e_{i'}^\sigma \quad = \quad MAX_\sigma(e_j^\sigma,\ \forall e_j : e_j \in \Sigma, e_j \neq e_i) \tag{3}$$

where MAX_σ denotes the statistical maximum operator which returns the statistical maximum of a set of random variables. Also, $e_{i'}$ is a fictional edge with path delay $e_{i'}^\sigma$.

Definition II.6 (**Critical Path**)**.** A critical path of a circuit implemented on a silicon die is the path which determines the maximum circuit delay. With process variations, different paths can be critical on different dies. Therefore, in a probabilistic scenario, every path of a circuit timing graph, G, has a certain probability of being the critical path.

Definition II.7 (**Local Criticality**)**.** The local criticality, τ_{ij}, of edge e_i with respect to e_j, is defined as the probability that at least one path of timing graph G passing through edge e_i takes on a value greater than or equal to any path passing through edge e_j, over all manufactured dies. The local criticality is given by the tightness probability in [13] and computed as,

$$\tau_{ij} \quad = \quad \Phi(\tfrac{\mu_i - \mu_j}{\theta}) \tag{4}$$

$$\text{with } \theta = \sqrt{\sigma_i^2 + \sigma_j^2 - 2 \cdot \rho_{ij} \cdot \sigma_i \cdot \sigma_j} \quad (5)$$

Here Φ is the cumulative distribution function (*cdf*) of a unit normal random variable, $\mathcal{N}(0,1)$. The mean and standard deviation of an edge, e_i, is given by μ_i and σ_i respectively. The correlation between edges e_i and e_j is given by ρ_{ij}.
Local criticality τ_{ij} can be thought of as the **degree of domination** of edge e_i over e_j. It is easy to see that $\tau_{ji} = 1 - \tau_{ij}$.

Definition II.8 (Global Criticality). The global criticality T_i (also referred to as criticality hereon) of edge e_i in cutset Σ is the probability that it has maximum delay among all the edges in the cutset, i.e.,

$$\begin{aligned} T_i &= \Pr(e_i^\sigma \geq e_j^\sigma) \quad \{\, \forall\, e_j \in \Sigma, \quad e_j \neq e_i \,\} \\ &= \Pr(e_i^\sigma \geq e_{i'}^\sigma) \quad \text{(see Def. II.5)} \\ &= \tau_{ii'} \quad \text{(from Eq. 4)} \end{aligned} \quad (6)$$

In other words, T_i represents the probability that at least one path passing through e_i has a delay greater than any other path not passing through it, over all the manufactured dies. T_i is also referred to as the **criticality probability** of e_i.

Physically, the local criticality of an edge e_i in a cutset corresponds to a comparison of its path delay with respect to another edge of the cutset whereas the global criticality of an edge corresponds to a comparison of its path delay with respect to all other edges in the cutset. It follows that the global criticality of $e_i \in \Sigma$ cannot be greater than its local criticality with respect to any other edge in Σ, i.e.,

$$T_i \leq \tau_{ij} \quad \{\, \forall\, e_j \in \Sigma, \quad e_j \neq e_i \,\} \quad (7)$$

Definition II.9 (MAX_θ). The statistical maximum, MAX_θ, of two normal random variables, x and y, in canonical form (see Eq. 2) using Clark's formulation [3], is given by $z = MAX_\theta(x,y)$, where ϕ is the normalized Gaussian probability density function (*pdf*), $\mathcal{N}(0,1)$, θ is defined in Eq. 5 and

$$\begin{aligned} \mu_z &= \tau_{xy} \cdot \mu_x + \tau_{yx} \cdot \mu_y + \theta \cdot \phi\left(\tfrac{\mu_x - \mu_y}{\theta}\right) \\ \sigma_z^2 &= \tau_{xy} \cdot (\sigma_x^2 + \mu_x^2) + \tau_{yx} \cdot (\sigma_y^2 + \mu_y^2) \\ &\quad + (\mu_x + \mu_y) \cdot \theta \cdot \phi\left(\tfrac{\mu_x - \mu_y}{\theta}\right) - \mu_z^2 \\ a_{zi} &= \tau_{xy} \cdot a_{xi} + \tau_{yx} \cdot a_{yi} \\ \zeta_z &= \sqrt{(\tau_{xy} \cdot \zeta_x)^2 + (\tau_{yx} \cdot \zeta_y)^2} \end{aligned} \quad (8)$$

Here μ_z and σ_z are the mean and standard deviation of z respectively and a_{zi} is the computed sensitivity to principal component, p_i, in the canonical representation of z (defined in Def. II.4). A weighting factor is applied to a_{zi} and the random component ζ_z to equate the variance of z to σ_z^2.

Note that MAX_θ is a linear approximation of the maximum of two Gaussian random variables as another Gaussian and is a particular implementation of the general statistical maximum operator MAX_σ. For the rest of the paper it is assumed that the time taken to compute τ_{xy} is similar to that taken to compute $MAX_\theta(x,y)$.

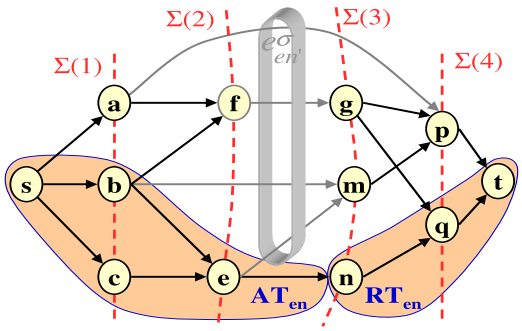


Fig. 1. Example timing graph, G with cutsets, $\Sigma(1) - \Sigma(4)$. The shaded portions are nodes and edges used to compute the arrival and required time of edge e_{en}. The complementary path delay of e_{en} is the statistical maximum delay of all paths not passing through it, i.e., $e_{en'}^\sigma = MAX_\sigma(e_{em}^\sigma, e_{bm}^\sigma, e_{fg}^\sigma, e_{ap}^\sigma)$.

III. Statistical Criticality Computation

We mention at the outset that our idea to compute edge criticalities in a timing graph, $G(V,E)$, is similar to [15], in which the notion of cutsets is used. Although both algorithms asymptotically take time linear in the number of edges in G, we use a zone-based approach. Section III-A illustrates the cutset computation procedure, followed by outlining a simple statistical criticality algorithm, BSC, with runtime complexity quadratic in the number of edges in G, in Section III-B. The runtime complexity is reduced in Section III-C using linear book-keeping data structures. The authors in [15] use a binary tree like data structure for this purpose. Sections III-D and III-E detail our zone-based approach, which reduces the criticality computation runtime complexity to linear in the number of edges in G. In [15], an array based structure is used for this purpose. The primary advantage of our method is that the cutsets can be processed in any order, i.e., to compute the criticality of an edge in a cutset we need not compute the edge criticalities in any of its predecessor or successor cutsets.

A. Cutset Computation

Fig. 1 illustrates the computation of cutsets on a timing graph $G(V,E)$. We topologically order the nodes in G from the virtual source to virtual sink node, followed by grouping them according to levels in G, such that nodes with a level lower than l are predecessors and nodes with a level greater than l are successors of nodes at level l, denoted $\Sigma_n(l)$. For instance, $\Sigma_n(2) = \{n_e, n_f\}$. Edges crossing l are denoted by $\Sigma_e(l)$ and these are called mc-edges.

Definition III.1 (mc-edge). An mc-edge is an edge with end-level at least two greater than its start-level.

Thus, with our level enumerated cutsets, these are edges which cross over at least one cutset. In Fig. 1, e_{ap} and $e_{bm} \in \Sigma_e(2)$ are a set of mc-edges crossing level 2. However, edges like e_{fg} and e_{en} are not mc-edges since they start at level 2 and end at level 3 with no cross over. Consider the set of nodes and edges given by,

$$\Sigma(l) = \Sigma_n(l) \cup \Sigma_e(l) \quad (9)$$

Algorithm 1 BSC ($G(V,E)$) // G = circuit timing graph

1: Perform a forward and reverse SSTA on G
2: Topologically order G and find its cutsets Σ
3: **for all** cutsets $\Sigma \in G$ **do**
4: **for all** edges $e_i \in \Sigma$ **do**
5: Compute $e_{i'}^{\sigma}$ // see Def. II.5
6: $T_i = \tau_{ii'}$ // see Def. II.8
7: **end for**
8: **end for**

By Def. II.2, $\Sigma(l)$ forms a cutset since every v_s to v_t path in G must pass through at least one member of $\Sigma(l)$ and its elements are disjoint. Our aim is to compute the criticalities of all edges in G, using Def. II.8. The topological level-enumerated cutsets are necessary and sufficient because they cover all the nodes and edges in G and no cutset is fully contained in another cutset. The number of such cutsets equals the number of levels L in G. To compute the criticality of all edges in G, we substitute nodes in Σ_n with their fanout edges. For instance, at level 2, we obtain the cutset of edges $\Sigma(2) = \{e_{ap}, e_{fg}, e_{bm}, e_{em}, e_{en}\}$.

B. BSC: Basic SC Algorithm

The simplistic approach called BSC, shown in Algorithm 1, computes global criticalities of all edges in timing graph G. Step 1 performs a forward and reverse SSTA to compute path delays (see Def. II.4) of all edges (nodes) in G followed by topologically ordering G into levels to compute cutsets, Σ. Steps 3-8 compute the criticality of an edge in Σ by first computing its complementary path delay and then using Eq. 6.

Due to the presence of mc-edges, each cutset, Σ, potentially contains $O(E)$ edges, where E is the number of edges in the timing graph. Moreover, since Step 5 computes the complementary path delay of an edge in time linear in the size of Σ, this step takes quadratic time, $O(E^2)$ over all edges in Σ. Over all cutsets in G, Algorithm 1 therefore has a complexity of $O(L \cdot E^2)$. Sections III-C and III-D discuss methods to reduce the time complexity of the basic approach.

C. Linear Time Book-keeping

From the previous discussion, computing the complementary path delay of all edges in a cutset takes quadratic time. Fig. 2 shows a cutset, $\Sigma = \{e_1, e_2, \ldots, e_5\}$. To compute the complementary path delay of edges e_1 and e_2, we compute

$$\begin{aligned} e_{1'}^{\sigma} &= MAX_{\sigma}(e_2^{\sigma}, e_3^{\sigma}, e_4^{\sigma}, e_5^{\sigma}) \\ e_{2'}^{\sigma} &= MAX_{\sigma}(e_1^{\sigma}, e_3^{\sigma}, e_4^{\sigma}, e_5^{\sigma}) \end{aligned} \quad (10)$$

Clearly, $MAX_{\sigma}(e_3^{\sigma}, e_4^{\sigma}, e_5^{\sigma})$ is a common term in Eq. 10 above. To speed up the computation of the complementary path delay of edges in a cutset, our book-keeping ordered lists aim to keep track of this common information.

Definition III.2 (Ordered Lists). Given an arbitrary set $\Sigma = \{e_1, e_2, \ldots, e_n\}$ of n random variables, we define forward and reverse ordered lists, denoted Υ_F and Υ_R respectively, as

$$\Upsilon_F(i) = MAX_{\sigma}(e_1^{\sigma}, \ldots, e_i^{\sigma}) \quad (11)$$

$$\Upsilon_R(i) = MAX_{\sigma}(e_i^{\sigma}, \ldots, e_n^{\sigma}) \quad (12)$$

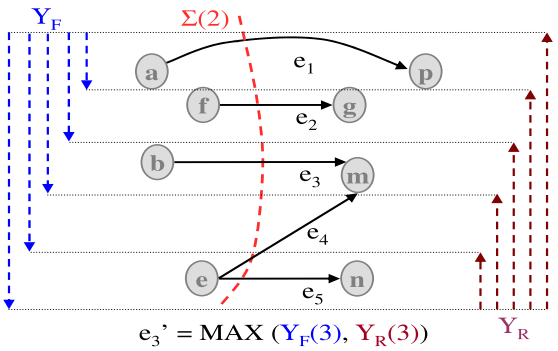


Fig. 2. Illustration of forward and reverse linear book-keeping data structures, Υ_F and Υ_R, to compute the complementary path delay of e_3 in cutset, $\Sigma(2) = \{e_{ap}, e_{fg}, e_{bm}, e_{em}, e_{en}\}$, from Fig. 1, with edges relabeled e_1, e_2, e_3, e_4, e_5 respectively.

The global criticality of an edge $e_i \in \Sigma$ (Def. II.8) can now be computed as

$$\begin{aligned} T_i &= \Pr\left(e_i^{\sigma} \geq MAX_{\sigma}(e_1^{\sigma}, \ldots, e_{i-1}^{\sigma}, e_{i+1}^{\sigma}, \ldots, e_n^{\sigma})\right) \\ &= \Pr\left(e_i^{\sigma} \geq MAX_{\sigma}(\Upsilon_F(i-1), \Upsilon_R(i+1)\right) \end{aligned} \quad (13)$$

Computation of Υ_F and Υ_R takes $2n$ MAX_{σ} operations. Eq. 13 takes two MAX_{σ} operations for a total of $4n$ MAX_{σ} operations over all edges in Σ. The ordered lists help compute criticalities of cutset edges in $O(n)$ time, i.e., linear in the size of the cutset. Fig. 2 illustrates this computation for edge e_3.

D. Zone Computation

Using Υ_F and Υ_R, Steps 5-6 of Algorithm 1 now take $O(E)$ as compared to $O(E^2)$ time. Typical circuits however contain many mc-edges (Def. III.1), as a result of which every cutset potentially has $O(E)$ edges. Over all L cutsets, we could take $O(L \cdot E)$ time, still a considerable slowdown.

To see why we can do better, example timing graph G in Fig. 3, depicts mc-edges e_1, e_2, e_3, e_4, e_5 and e_6. Consider traversing G to compute the criticality of its edges using Algorithm 1. To compute the criticality of an edge, say e_x, at level 1, we compute its complementary path delay, $e_{x'}^{\sigma}$, which includes $MAX_{\sigma}(e_1^{\sigma}, e_2^{\sigma})$. At level 2, for $e_{y'}^{\sigma}$, we compute $MAX_{\sigma}(e_1^{\sigma}, e_2^{\sigma}, e_3^{\sigma}, e_4^{\sigma})$. Clearly, the only information needed at level 2 with regard to edges e_1 and e_2 is $MAX_{\sigma}(e_1^{\sigma}, e_2^{\sigma})$, already computed at level 1. Algorithm 1 redundantly re-computes this information at level 2, thus accounting for the multiplicative L factor in the computation cost. The basic idea of zones is to abstract out the maximum of the mc-edges thereby reusing information to the greatest possible extent.

Let us reconsider traversing the timing graph in Fig. 3. At level 1 we would like to forward accumulate the maximum of mc-edges e_1 and e_2, used to compute the criticality of edges at higher levels. We thus enter an accumulation phase beginning at level 1, to obtain $Z_{1F} = \{e_1, e_2\}$ and $Z_{1F}^{\sigma}(1) = MAX_{\sigma}(e_1^{\sigma}, e_2^{\sigma})$, useful at level 2 to find the maximum of the mc-edges crossing it (to compute $e_{y'}^{\sigma}$). At level 2 we accumulate edges e_3 and e_4 to obtain $Z_{1F} = \{e_1, e_2, e_3, e_4\}$ and $Z_{1F}^{\sigma}(2) = MAX_{\sigma}(Z_{1F}^{\sigma}(1), e_3^{\sigma}, e_4^{\sigma})$. The indices of Z_{1F}^{σ}, denoted z_{1F}, are time points recording the order in which

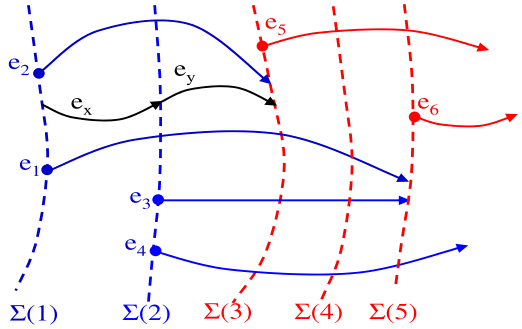


Fig. 3. An example timing graph, G, with mc-edges, $\Sigma_e = \{e_1, e_2, \ldots, e_6\}$. Edges e_x and e_y are not mc-edges.

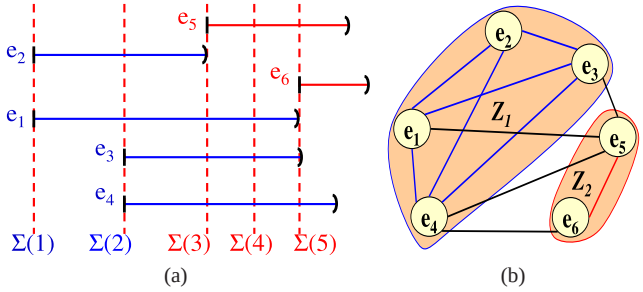


Fig. 4. Fig. 4(a) shows the mc-edges, $\Sigma_e = \{e_1, \ldots, e_6\}$, of the timing graph in Fig. 3, represented as half-open intervals. Fig. 4(b) shows the corresponding interval graph representation, G_e, with zones, $Z_1 = \{e_1, e_2, e_3, e_4\}$ and $Z_2 = \{e_5, e_6\}$, identified as mutually exclusive maximal cliques.

mc-edges accumulate in Z_{1F}. At level 3 however, we can no longer accumulate e_5 in Z_{1F}, since e_2 has reached its end level and does not contribute to the accumulated maximum in Z^{σ}_{1F}. Z_{1F} is thus a maximal set representing all edges accumulated in phase 1. Note that at this point, Z^{σ}_{1F} is not useful to us.

We now begin a reverse accumulation phase, to compute the order in which edges belonging to Z_{1F} leave timing graph, G. Once again, this is precomputed by a traversal of G as, $Z_{1R} = \{e_1, e_3, e_4\}$ and $Z^{\sigma}_{1R}(1) = MAX_{\sigma}(e^{\sigma}_1, e^{\sigma}_3, e^{\sigma}_4)$ at time point 1. Similarly at time point 2, $Z_{1R} = \{e_4\}$ and $Z^{\sigma}_{1R}(2) = e^{\sigma}_4$. The indices of Z^{σ}_{1R}, denoted z_{1R}, are time points that record the order in which mc-edges leave Z_{1R}. We concurrently start a new accumulation phase beginning with edge e_5 as $Z_{2F} = \{e_5\}$ and $Z^{\sigma}_{2F}(1) = e^{\sigma}_5$. The maximum of mc-edges crossing levels greater than 3 (for example at level 4), can now be computed using $MAX_{\sigma}(Z^{\sigma}_{1R}(1), Z^{\sigma}_{2F}(1))$.

Definition III.3 (Zone). A zone Z_i is a set of mc-edges with the end-level of any edge higher than the start-level of all edges in Z_i, i.e., edges enter a zone before any edge exits it.

From the above description, at a particular level, l, of the timing graph, the different mc-edges that cross it can be active in different zones. At level l, the contribution, Z^{σ}_{iMAX}, of mc-edges belonging to zone Z_i, is given respectively by the appropriately indexed entry of Z^{σ}_{iF} or Z^{σ}_{iR}, depending on the forward or reverse accumulation phase of the zone. The statistical maximum of mc-edges crossing level l, denoted Z^{σ}_{MAX}, is obtained by computing the maximum of the contributions of each zone, Z^{σ}_{iMAX}, over all the zones.

Formally, mc-edges represent half-open intervals, from their source to sink level, as in Fig. 4(a), denoted Σ_e, with the corresponding interval graph representation shown in Fig. 4(b), denoted G_e. The interval graph is a one-to-one representation of intervals to vertices, with two vertices connected by an edge if and only if their corresponding intervals overlap [4]. In what follows, the term interval is used interchangeably with edge.

By Def. III.3, a zone is any set of overlapping intervals. In the interval graph representation, a zone is a clique (not necessarily maximal). Hence, like [16], we aim to compute the cliques in the interval graph. Since an mc-edge belongs to a single zone, the cliques must be mutually exclusive. Fig. 4(b) shows one set of mutually exclusive cliques which forms the zones. We begin with edge e_2 and greedily compute the maximal clique $\{e_1, e_2, e_3, e_4\}$ to form zone Z_1. Next, with e_5 we get zone Z_2 with maximal clique $\{e_5, e_6\}$.

In the worst case, the number of zones, K, in a timing graph with L levels is $O(L)$. The idea is to minimize K so as to reduce the number of MAX_{σ} operations over all zones to compute Z^{σ}_{MAX}. The Algorithm described in [6] computes a minimum clique covering of an interval graph, G_e. A simplicial vertex of G_e is defined as follows.

Definition III.4 (Simplicial edge). A simplicial vertex, v^s, of an interval graph, G_e, is a vertex, all of whose neighbors form a clique with v^s [4]. The interval e^s in the corresponding interval representation, Σ_e, of G_e is called a simplicial edge. It is easy to verify that an interval with minimum end-level is a simplicial edge in Σ_e. In fact, the neighbors of v^s form a clique which is maximal. In Fig. 4, e_2 is a simplicial edge.

Algorithm 2, like [6], finds a minimum size clique covering of the interval graph, G_e, by repeatedly finding a simplicial edge in Σ_e, and removing all the edges overlapping it, i.e., it repeatedly computes mutually exclusive maximal cliques in G_e. These cliques form the zones in our criticality computation algorithm. However, unlike [6], a separate step to sort the intervals according to their end points is not needed, because of the topological ordering of the timing graph described in Section III-A. For the example interval representation in Fig. 4, we compute $e_2 \in \Sigma_e$ as the first simplicial edge, with zone $Z_1 = \{e_1, e_2, e_3, e_4\}$, followed by $e_5 \in \{\Sigma_e - Z_1\}$, as the second simplicial edge, with zone $Z_2 = \{e_5, e_6\}$.

Algorithm 2 computes zones linearly traversing timing graph, G, from the virtual source to virtual sink node, identifying mc-edges and reporting the mutually exclusive maximal cliques by keeping track of when an edge enters (Steps 13-16) and leaves (Steps 4-12) G. The zones are computed as $Z = \{Z_1 \cup Z_2 \cup \ldots \cup Z_K\}$, and the claim is that K is the minimum number of zones (cliques) needed to cover Σ_e (the interval graph, G_e). The following property proves this claim.

Property III.1. In Algorithm 2, the first edge, e^s_j, to exit its zone, Z_j, is a simplicial edge of the intervals corresponding to $\Sigma_e - \{Z_1 \cup Z_2 \cup \ldots \cup Z_{j-1}\}$.

Proof: Step 7 computes the first edge, e^s_j, to exit its zone,

Algorithm 2 Z = ComputeZones (Σ_e)
// Σ_e = mc-edges in the timing graph organized as per levels
// Z = list of mutually exclusive zones, $\{Z_1, \ldots, Z_K\}$

1: $K = 1$, $Z = \{\ \}$ // initialize the list of zones
2: $Z_K = \{Z_{KF} = \{\ \}, Z_{KR} = \{\ \}\}, z_{KF} = z_{KR} = 0$
// Z_{KF} (Z_{KR}) records the history of edges entering
// (leaving) zone Z_K, indexed by pointer z_{KF} (z_{KR})
3: **for all** levels $l \in G$ **do**
4: **for all** $e_j \in \Sigma_e$ with end level l **do**
5: Z_j = zone of e_j
6: Insert e_j into Z_{jR}, $++z_{jR}$ // e_j exits zone Z_j
7: **if** $Z_j == Z_K$ **then** // $e_j^s = e_j$ is the first to exit Z_j
8: $++K$ // create new zone
9: $Z_K = \{Z_{KF} = \{\}, Z_{KR} = \{\}\}, z_{KF} = z_{KR} = 0$
10: Insert Z_K into Z
11: **end if**
12: **end for**
13: **for all** $e_i \in \Sigma_e$ with start level l **do**
14: Set zone of e_i to Z_K
15: Insert e_i into Z_{KF}, $++z_{KF}$ // e_i enters zone Z_i
16: **end for**
17: **end for**
// Compute book-keeping lists for all zones
18: **for all** $Z_i \in Z$ **do**
19: Compute Υ_{iF} (Υ_{iR}) for Z_{iF} (Z_{iR}) // Eq. 11 (12)
20: **end for**
21: **return** Z

Z_j. Edge e_j^s has minimum end-level, l_j, in $\Sigma_e - \{Z_1 \cup Z_2 \cup \ldots \cup Z_{j-1}\}$ and therefore is a simplicial edge in $\Sigma_e - \{Z_1 \cup Z_2 \cup \ldots \cup Z_{j-1}\}$. If this was not the case, consider another edge, e_n, with end-level, $l_n < l_j$, $e_n \notin \{Z_1 \cup Z_2 \cup \ldots \cup Z_{j-1}\}$. Clearly, $e_n \notin Z_k$, $k > j$, because edges are assigned to zones in sequence implying its start-level (and thereby its end-level, l_n) must be greater than l_j. Therefore $e_n \in Z_j$, which is again a contradiction since it implies $l_n \geq l_j$. ■

Using Property III.1, Algorithm 2 like [6] repeatedly finds simplicial edges in Σ_e to compute a minimum size clique cover of its corresponding interval graph representation, G_e.

Lists Z_{iF} and Z_{iR} record the history of mc-edges entering and exiting zone Z_i, indexed by pointers z_{iF} and z_{iR} respectively (Steps 15 and 6). For the example in Fig. 4(a), the lists for zones Z_1 and Z_2 are computed as,

$$\begin{array}{lllllll} Z_{1F} & = & \{e_1, e_2, e_3, e_4\} & Z_{2F} & = & \{e_5, e_6\} \\ Z_{1R} & = & \{e_4, e_3, e_1, e_2\} & Z_{2R} & = & \{e_6, e_5\} \end{array} \quad (14)$$

For each zone Z_i, Step 19 computes the forward (Υ_{iF}) and reverse (Υ_{iR}) book-keeping lists, for Z_{iF} and Z_{iR} respectively.

In terms of computational complexity, Steps 3-17 of Algorithm 2 process each edge in Σ_e twice, first at its start-level and then at its end-level. Step 19 computes the forward and reverse book-keeping lists for the mutually exclusive zones in $O(|Z_1|)+O(|Z_2|)+\ldots+O(|Z_K| = O(|\Sigma_e|)$ time, where $|Z_i|$ is the number of edges in each zone and K is the minimum number of zones to cover the mc-edge interval representation, Σ_e, of the timing graph. Overall, the runtime of Algorithm 2 is $O(|\Sigma_e|)$, i.e., linear in the number of mc-edges.

It should be noted that as shown in [6], this approach is optimal, i.e., the lower bound on the computational complexity of computing a minimum clique cover is $\Omega(|\Sigma_e| \log |\Sigma_e|)$, if the mc-edges are not sorted by their end points.

Algorithm 3 ZSC ($G(V, E)$) // G = circuit timing graph

1: Perform a forward and reverse SSTA on G
2: Topologically order G and find its cutsets Σ
3: Compute Σ_e, the set of mc-edges
4: Z = ComputeZones(Σ_e)
5: **for all** levels $l \in G$ **do**
6: **for all** $e_j \in \Sigma_e$ with end-level l **do**
7: $++z_{jR}$ // e_j exits zone Z_j, update reverse pointer
8: $Z_{jMAX} = \Upsilon_{jR}(z_{jR})$
9: **end for**
10: **for all** $e_i \in \Sigma_e$ with start-level l **do**
11: $++z_{iF}$ // e_i enters zone Z_i, update forward pointer
12: $Z_{iMAX} = \Upsilon_{iF}(z_{iF})$
13: **end for**
14: $Z^{\sigma}_{MAX} = -\infty$ // maximum over all active zones
15: **for all** $Z_k \in Z$ **do**
16: $Z^{\sigma}_{MAX} = MAX_{\sigma}(Z^{\sigma}_{MAX}, Z^{\sigma}_{kMAX})$
17: **end for**
18: Create fictional edge Z_{MAX} with path delay Z^{σ}_{MAX}
19: $\Sigma = \{$Fanout edges of the nodes in $\Sigma_n(l)\} \cup Z_{MAX}$
// see Section III-A
20: Create Υ_F and Υ_R for Σ // see Section III-C
21: Compute $T_i\ \forall\ e_i \in \Sigma$ // see Def. II.8
22: **end for**

E. ZSC: Zone Based SC Algorithm

Our zone-based criticality computation technique, ZSC, is shown in Algorithm 3. Step 4 computes zones in timing graph G, in time linear in the size of Σ_e, the set of mc-edges. We then forward traverse G from v_s to v_t. Steps 6-13 update the forward and reverse history pointers of each zone, to compute the contribution, Z^{σ}_{iMAX}, of the mc-edges belonging to zone, Z_i, in constant time. Steps 14-17 compute Z_{MAX}, a fictional edge representing the statistical maximum, Z^{σ}_{MAX} of mc-edges crossing a particular level, depending on the contribution, Z^{σ}_{iMAX}, to each zone Z_i. Since we have on the order of $O(L)$ number of zones, over all levels of G, this step takes $O(L^2)$ time. Finally, using the book-keeping ordered lists from Section III-C, we compute global criticalities of edges in cutset Σ, in time linear in the number of edges in Σ. The overall runtime of the ZSC algorithm is therefore $O(E + L^2)$, which for a reasonably sized practical circuit is $O(E)$.

In summary, like [15], the zone-based approach computes the criticality of edges in timing graph $G(V, E)$, with a linear runtime complexity $O(E)$. Although both algorithms asymptotically take $O(L)$ time to compute the MAX_{σ} of mc-edges crossing a level, Algorithm 2 computes a minimum clique cover and helps to reduce the total number of MAX_{σ} operations computed in Steps 14-17 over all the cutsets. More importantly, our algorithm can compute the criticality of edges in a cutset, independently of other cutsets.

TABLE I
COMPARISON OF MONTE CARLO AND MAX_θ FOR THE *abc* PROBLEM

Method	T_a	T_b	T_c
MC	0.923	0.000	0.077
Clark	0.356	0.297	0.079
% Error δ	56.7%	29.7%	0.2%

IV. ERRORS IN ZSC

We ran Algorithm 3 on a subset of the ISCAS89 benchmarks to compute the global criticalities of all edges in the timing graph, G. We compared our implementation with a Monte Carlo (MC) simulation of 10000 samples and noted the absolute maximum difference in the criticalities of edges (denoted δ hereon). The difference was larger than 50% (for example, an edge reported by MC as 80% critical was reported by Algorithm 3 as 30% critical). In the following sections, we illustrate the sources of these errors with 3 random variables in a simple example we call the *abc* problem.

A. The abc Problem

As an illustration of these errors, consider a cutset Σ with random variables a, b and c, each with independent principal components (PCs) p_1 and p_2 (where p_i is a unit normal Gaussian, $\mathcal{N}(0,1)$), shown below,

$$\begin{aligned} a &= 4.000 + 0.5000 \cdot p_1 + 0.5000 \cdot p_2 \\ b &= 3.999 + 0.4999 \cdot p_1 + 0.5001 \cdot p_2 \\ c &= 3.800 + 0.6001 \cdot p_1 + 0.3999 \cdot p_2 \end{aligned} \quad (15)$$

It can be observed that a and b are nearly identical highly correlated random variables, and for any sample value of the $p_i's$, $a \geq b$ (high correlation coupled with the difference in means ensures that $\Pr(b \geq a) \approx 0.0$).

We ran a MC simulation with 100000 samples to determine the global criticalities of a, b and c. Table I shows a comparison with Clark's formulation, MAX_θ (see Def II.9). The columns $T_i,\ i \in \{a, b, c\}$, depict the global criticality of variable i. As seen in the last row of Table I, errors of 57% in the global criticality of a and 30% in that of b were observed.

B. Local and Global Errors

For a better illustration of the *abc* problem, Fig. 5 depicts the scenario of Eq. 15, using just one PC, p. We make the following observations.

1) The local criticality of b with respect to a, i.e., $\tau_{ba} \approx 0$. This is indicated by a large value of $\gamma \gg 3\sigma_p$ (the region where $b \geq a$). Moreover, Clark's tightness probability formulation from Eq. 4 also gives $\tau_{ba} \approx 0$.
2) Global criticality of b, $T_b \approx 0$. This is evident in Fig. 5 where regions $a \geq MAX_\theta(b, c)$ and $c \geq MAX_\theta(a, b)$ cover the entire probability space.

Observations 1 and 2 are consistent with Eq. 7. Now consider computing the global criticality of b, using the cutset approach. We first compute its complementary path delay $b' = MAX_\sigma(a, c)$. It follows from Def. II.8 that $T_b = \Pr(b \geq MAX_\sigma(a, c))$. With Clark's formulation MAX_θ, for the statistical MAX_σ, we get $T_b = \Pr(b \geq MAX_\theta(a, c)) = 0.297$.

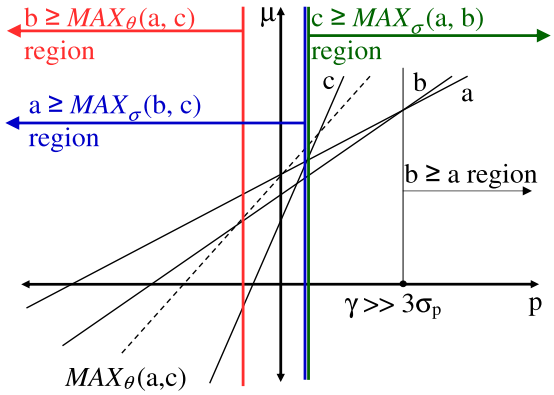


Fig. 5. A pictorial depiction (not to scale) of the *abc* example with random variables a, b and c with one PC, p.

Intuitively, for this scenario, Clark's formulation is accurate with respect to local criticality t_{ba} of b, but it overestimates its global criticality T_b, and is inconsistent with Eq. 7.

Definition IV.1 (Local Errors). With respect to Clark's formulation, edge e_i in cutset Σ is said to have **local errors** iff there exists some edge $e_j \in \Sigma$ with respect to which its local criticality is less than its global criticality, i.e.,

$$\{\ \exists\ e_j\ \in\ \Sigma,\quad e_j \neq e_i\ \}\ :\ \tau_{ij}\ <\ T_i \quad (16)$$

In other words, Eq. 7 does not hold. By definition, local errors always overestimate the criticality of an edge in Σ. In our toy example, b exhibits local errors of magnitude 0.297, with respect to a. Local errors were found to propagate in the ZSC algorithm, where variables (edges) like b that should not have been critical, were found to have a significant criticality.

It must be pointed out that as was shown in [12], the order of variables plays an important role due to Clark's MAX_θ approximation. For the *abc* problem however, ordering variables (a and c) in the MAX_θ operation will not eliminate local errors in b. Local errors are an artifact of the manner in which we compute global criticalities.

Local errors only present a part of the picture with respect to the overall errors seen in criticality computation. This is because of the inherent inconsistencies in using Clark's formulation, MAX_θ, to approximate the maximum of a set of Gaussian random variables as another Gaussian. The works in [12] and [17] for instance, give a detailed analysis of the errors involved in such an approximation.

Definition IV.2 (Global Errors). With respect to Clark's formulation, edge e_i in a cutset Σ is said to have **global errors**, iff its computed criticality T_i differs from its true criticality and the edge does not exhibit local errors, i.e.,

$$T_i\ \leq\ \tau_{ij}\ \{\ \forall e_j \in \Sigma, e_j \neq e_i\ \}\ \text{and}\ T_i\ \text{in error.} \quad (17)$$

Global errors cause erroneous values of the global criticality of an edge, e_i, in a cutset, due to the inaccuracies in the computation of its complementary path delay, e_i^σ, using Clark's approximation. For the *abc* example, T_a is underestimated by

0.567. Note that the value of T_a is consistent with Eq. 7, since both $\tau_{ab} = 1.0$ and $\tau_{ac} = 0.921$ are greater than $T_a = 0.356$.

Two observations motivate the need for the pruning based criticality algorithm, described in Section V. First, with respect to local errors in variable b, if we choose to "ignore" variable c and compute the criticality of b directly with respect to a, we get, $T_b = \tau_{ba} = 0.0$, a better result, since a almost completely dominates b. Second, with respect to global errors in variable a, if we choose to "ignore" variable b due to its high dominance with respect to a and compute the criticality of a directly with respect to c, we get, $T_a = \tau_{ac} = 0.921$, a better result, since the computation of $MAX_\theta(b, c)$ (and hence the inaccuracy involved in it) is avoided.

In summary, although local and global errors result from Clark's MAX_θ linear approximation, local errors are an artifact of the manner in which we compute global criticalities of edges in a cutset whereas global errors are more fundamental, arising due to the inherent approximation of MAX_θ.

V. Clustering Based Statistical Criticality Computation

Definition V.1 (Dominant and Non-dominant Edges). An edge, e_i, in set, Σ, is **dominant** iff its local criticality with respect to all other edges in Σ is above a threshold ε, i.e.,

$$\tau_{ij} \quad > \quad \varepsilon \quad \{ \forall\, e_j \ \in \ \Sigma, \quad e_j \neq e_i \ \} \tag{18}$$

Otherwise, edge e_i is said to be **non-dominant** in Σ, i.e.,

$$\{ \exists\, e_j \ \in \ \Sigma, \quad e_j \neq e_i \ \} \ : \ \tau_{ij} \quad \leq \quad \varepsilon \tag{19}$$

Definition V.2 (Mutually-dominant Edges). A set, Σ, of edges are said to be **mutually dominant** iff each edge in Σ is dominant, i.e.,

$$\tau_{ij} \quad > \quad \varepsilon \quad \{ \forall\, e_i, e_j \ \in \ \Sigma, \quad e_j \neq e_i \ \} \tag{20}$$

As seen in the previous section, non-dominant edges (like b in Fig. 5) in a cutset exhibit local errors. Moreover, they also contribute to global errors of other edges in the cutset (like a in Fig. 5). To avoid the bulk of these errors, we propose to prune the cutset, eliminating its non-dominant edges from injecting errors in global criticality computations.

Pruning is justified by Eq. 7, wherein eliminating edge e_i with local criticality lower than a sufficiently small threshold value ε does not hurt global criticality computations because $T_i \leq \varepsilon$. The benefits are accentuated in cutsets with dominant edges that have large global criticalities, since the sum of global criticalities across a cutset must equal 1.0 (implying that many edges have very small local criticalities).

However, not every edge with global criticality below ε can be eliminated by pruning, particularly if its local criticality is greater than ε. Such edges cause global errors in the cutset.

A. ${}^nC_2$ Cutset Pruning

A straightforward approach to prune a cutset would be to perform a pairwise comparison of edges, eliminating those that have a local criticality less than a predefined threshold ε. The main drawback of this approach is its prohibitive quadratic runtime complexity of $O(n^2)$, due to ${}^nC_2$ local criticality computations, where n is the number of edges in the cutset.

Algorithm 4 K = KCenterPrune (Σ, ε, S)
// Σ = cutset of edges; ε = pruning threshold;
// S = maximum cluster size; K = # clusters

1: $\Omega = \{\ \}$ // set of clusters
2: $\sigma = \{\ \}$ // initialize the 1^{st} cluster
3: $K = 0$ // total number of clusters present in Ω
4: seed $\chi \in \Sigma$ = object (or edge) with maximum mean μ
5: Insert χ as the center of cluster σ
6: **for all** $i \in \Sigma$ **do**
7: **if** $\tau_{i\chi} > \varepsilon$ **then** // see Def. II.7
8: Insert i in σ // object i not dominated by χ
9: **end if**
10: **if** $\tau_{\chi i} \leq \varepsilon$ **then**
11: Mark χ = pruned // object χ dominated by i
12: **end if**
13: **end for**
14: Compute radius, r_σ and distal element, R_σ of σ
15: Insert cluster σ into Ω; $++K$
16: **while** (maximum size of a cluster in $\Omega > S$) **do**
17: σ = CreateNewCluster (Ω)
18: Insert new cluster σ into Ω; $++K$
19: **end while**
20: Insert all un-pruned objects of Ω in Σ and return K

B. Clustering Based Cutset Pruning and Ordering

To overcome the quadratic runtime complexity overhead of the aforementioned ${}^nC_2$ approach, we present a new clustering based pruning technique which uses the K-center clustering algorithm of [5].

The basic idea is to prune the non-dominating edges from the cutset to return a set of mutually dominant edges. Throughout the execution of the algorithm, a dominant edge, selected from the current set of edges, Σ, is used to prune out non-dominant edges from Σ. Clustering facilitates the selection of dominant edges. The variables used in the algorithm are:

- σ: A cluster containing at least one object.
- κ: Each cluster σ contains a center, κ.
- $d_{i\kappa}$: Distance of an object, i, from its cluster center, κ, is its local criticality, $\tau_{i\kappa}$, with respect to κ.
- r_σ: Radius of cluster σ, is the distance of the object farthest from center κ, i.e., $r_\sigma = max(d_{i\kappa})\ \forall i \in \sigma$.
- R_σ: A distal object of cluster σ is an object with maximal distance from κ, i.e., $R_\sigma = j \ : \ d_{j\kappa} = r_\sigma$. In case of multiple distal elements we choose one arbitrarily.

Algorithm 4 describes the procedure. We first choose the seed χ, as the object with maximum mean in cutset, Σ (Steps 4-5). Next, Steps 7-9 prune Σ with respect to seed, χ, also marking χ as pruned if its local criticality with respect to any other object in Σ is less than ε (Steps 10-12). Steps 16-19 iteratively compute new clusters from existing ones (Algorithm 5) until no cluster has size exceeding S. Step 20 returns the remaining un-pruned objects in Σ.

In Algorithm 5, the distal element, χ, of the cluster, m, with maximum radius is chosen as the center of a newly created cluster, σ (Steps 1-4). Intuitively, χ is the object upon which its center has the lowest degree of domination (Def II.7) and

Algorithm 5 σ = CreateNewCluster (Ω)
// Ω = set of clusters; σ = new cluster

1: $\sigma = \{\ \}$ // initialize new cluster σ
2: $m =$ cluster with maximum radius in Ω
3: $\chi = R_m$ // distal element of cluster m
4: Insert χ as center of newly created cluster σ
// Prune Ω with respect to χ
5: **for all** $j \in \Omega, j \neq \kappa, \kappa =$ center of a cluster in Ω **do**
6: **if** $\tau_{j\chi} < \varepsilon$ **then** // χ dominates j (see Def. II.7)
7: Delete j from Ω // prune j
8: **else if** $\tau_{j\chi} < d_{j\kappa}$ **then** // χ dominates j more than κ
9: Remove j from its current cluster, insert j in σ
10: **end if**
11: **end for**
12: **if** $\exists\ j \in \Omega\ :\ (1 - \tau_{j\chi}) \leq \varepsilon$ **then** // j dominates χ
13: Mark $\chi =$ pruned
14: **end if**
15: Compute r_σ and R_σ for σ and all existing clusters in Ω
16: **return** σ

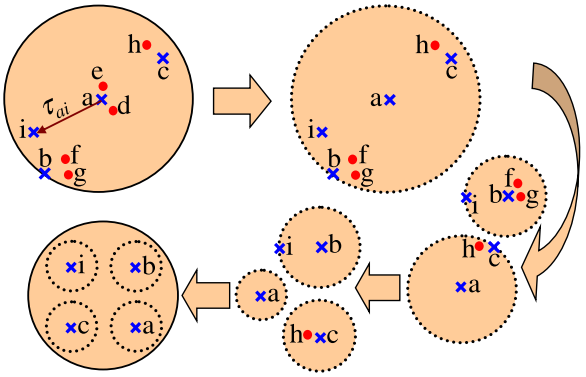


Fig. 6. Illustration of the clustering based pruning procedure of Algorithm 4. Crosses indicate dominant objects and dots indicate non-dominant objects. The clustering distance is the local criticality (τ_{ai}) of an edge (i) from its cluster center (a).

hence a good candidate to facilitate the pruning of other edges in the cutset. Therefore it is chosen as the center of the new cluster. Step 7 uses χ to prune objects j (with local criticality with respect to χ less than ε) from their respective clusters. If χ has a higher degree of domination over j compared to its current center κ, j is removed from its current cluster and inserted into new cluster σ (Steps 8-10). Intuitively, a greater degree of domination between two edges results in smaller global errors in MAX_θ. If the newly added cluster center χ, is dominated, it is marked pruned (Step 13).We return the newly created cluster σ after adjusting the radius and the distal element of all currently existing clusters in Ω (Steps 15-16).

Fig. 6 illustrates the execution of Algorithm 4 on a cutset of 9 objects labeled a-i with pruning threshold $\varepsilon = 0.05$, taken from one of the ISCAS89 benchmarks (s9234). The relevant local criticalities of the objects are, $\tau_{ba} = 0.19$, $\tau_{ca} = 0.18$, $\tau_{da} = 0.01$, $\tau_{ea} = 0.0$, $\tau_{fa} = 0.17$, $\tau_{ga} = 0.17$, $\tau_{ha} = 0.17$, $\tau_{ia} = 0.17$, $\tau_{fb} = 0.02$, $\tau_{gb} = 0.02$, $\tau_{ib} = 0.06$ and $\tau_{hc} = 0.03$. Initially, a is chosen as the center of the 1st cluster, pruning out objects d and e. Next, b, a distal element of cluster 1 becomes the center of cluster 2, pruning out objects f and g. Also, since $\tau_{ib} < \tau_{ia}$, object i is absorbed into cluster 2. Next c, the distal element of cluster 1, the cluster with maximum radius, is chosen as the center of cluster 3, pruning object h. Finally, object i becomes the center of cluster 4 and the algorithm returns mutually dominant objects a, b, c and i. The algorithm has the following properties.

Property V.1. At any iteration, all objects in Ω (excluding cluster centers marked pruned) are dominant with respect to all existing cluster centers.

Proof: To avoid being pruned, objects must be dominant with respect to seed χ, which is also the center of the 1st cluster (Step 8 of Algorithm 4). Moreover, every object j is compared with all newly added cluster centers in Line 7 of Algorithm 5. Clearly, any object j must be dominant with respect to these centers to avoid being pruned. Moreover, Lines 11 of Algorithm 4 and 13 of Algorithm 5 compare every cluster center with every object for dominance. Although not immediately removed from Ω, centers are marked pruned if they are non-dominant with respect to other cluster objects. ■

Property V.2. With $S = 1$, KCenterPrune(Σ, ε, 1) returns a set of mutually dominant edges (see Def.V.2) in Σ.

Proof: When $S = 1$, each cluster in Ω contains only one object, its cluster center. From Property V.1 above we know that these are either marked pruned or are dominant with respect to other cluster centers. It follows from step 20 of Algorithm 4 (which returns all un-pruned objects of Ω), Σ contains mutually dominant objects. ■

Property V.3. For any cluster $\sigma \in \Omega$, its center, χ, has a **higher degree of domination** over its members than any other cluster center κ, i.e.,

$$\tau_{\chi j} > \tau_{\kappa j} \quad \{\forall\ j \in \sigma,\ \kappa \in \Omega,\quad \kappa \neq \chi\} \tag{21}$$

Proof: This is evident from Steps 8-10 of Algorithm 5. Each object in Ω is compared with the new cluster center χ. The condition $\tau_{j\chi} < d_{j\kappa}$ is equivalent to $\tau_{\chi j} > \tau_{\kappa j}$, i.e., the new cluster center, χ, has a higher degree of domination over object j than its cluster center, κ. ■

Property V.4. For a cutset Σ of size n and K clusters returned, KCenterPrune takes $O(nK)$ time.

Proof: A single run of Algorithm 5 compares every object in Σ with center χ of the new cluster σ, taking $O(n)$ time. Since each iteration in Algorithm 4 returns a new cluster, with K clusters returned, the overall runtime is $O(nK)$. ■

C. CPSC: Clustering Based SC Algorithm

Algorithm 6 derives mainly from Algorithm 3 combined with Algorithm 4 to compute the statistical criticality (SC). The main difference is Steps 3-15 (differ from Steps 6-13 of Algorithm 3), which update the zone information, accounting for pruned edges in cutsets from previous levels. Unlike Algorithm 3, we only compute the contribution of an mc-edge, e_i, to its zone, Z_i, if it is un-pruned in previous levels. Therefore

Algorithm 6 CPSC $(G(V,E),\varepsilon)$
// G = circuit timing graph; ε = pruning threshold

1: Algorithm 3, Steps 1-4 to obtain a cutset Σ of edges
2: **for all** levels $l \in G$ **do**
3: **for all** $e_j \in \Sigma_e$ with end-level l **do**
4: **if** e_j is the first edge to exit zone Z_j **then**
5: Remove pruned edges from Z_{jR}; Recompute Υ_{jR}
6: **end if**
7: **if** e_j is un-pruned **then**
8: $++z_{jR}$ // e_j exits zone Z_j, update reverse pointer
9: $Z_{jMAX} = \Upsilon_{jR}(z_{jR})$
10: **end if**
11: **end for**
12: **for all** un-pruned $e_i \in \Sigma_e$ with start-level $l-1$ **do**
13: Z_i = zone of e_i
14: $Z^{\sigma}_{iMAX} = MAX_{\sigma}(Z^{\sigma}_{iMAX}, e^{\sigma}_i)$
15: **end for**
16: Algorithm 3, Steps 14-19 to compute cutset Σ
17: K = KCenterPrune $(\Sigma, \varepsilon, 1)$ // pruning Σ
18: KCenterPrune (Σ, ε, S) // ordering Σ
19: Algorithm 3, Steps 20-22 to compute the global criticality of all edges in the pruned Σ
20: **end for**

we do not need forward book-keeping data structure Υ_{iF}, to compute Z^{σ}_{iMAX}, the maximum of mc-edges belonging to Z_i, crossing the current level. Instead, Z^{σ}_{iMAX} is computed online, in Steps 12-15. Due to pruning, the computed reverse book-keeping data structure, Υ_{jR}, of a zone Z_j, may be invalid. On encountering the first edge leaving this zone, we recompute Υ_{jR}, removing all pruned edges from it (Steps 4-6). This is allowed because all edges enter a zone (and therefore it is known if they have been pruned) before any edge exits it.

Step 17 derives a set of mutually dominant edges from cutset Σ, facilitated using Property V.2. Step 18 orders cutset Σ, facilitated by Property V.3. There can be many different orderings when performing the statistical maximum of edges in the cutset [12]. Property V.3 proves that a cluster center has a higher degree of domination over its members than any other cluster center. Therefore, in the order of edges returned, an edge is closer to its most dominating center (as opposed to the case in which a purely random order were chosen). The intuition is that a greater degree of domination between two edges would result in smaller errors in the MAX_{θ} operation, as shown in [17]. Algorithm 4 stops execution when the maximum cluster size equals S. If S were set to a large number, like the size of the cutset, the algorithm would exit without any clustering iterations and a random ordering would result. For our experiments, we heuristically chose a cluster size S equal to the square root of the number of edges in the cutset, to balance out the number of edges in each cluster and help to reduce the runtime of the ordering step by performing a fewer number of iterations. Our framework is also flexible enough to accommodate other error metrics like [12] or the skewness. Such an ordering cannot be obtained with the ${}^nC_2$ pruning strategy of Section V-A. Section VI-A, discussed later, uses a sampling technique which obviates the need for the ordering step. Property V.4 ensures that in a cutset with n edges having a small number of dominant edges, K ($K \ll n$), Algorithm 4 runs in $O(n)$ or linear time.

In summary, our clustering based algorithm eliminates non-dominant edges from the cutset so as to reduce errors (due to Clark's maximum operation, MAX_{θ}) in the global criticality computation of the dominant edges. Ideally, computing the maximum operation accurately would significantly reduce errors in global criticality. Various techniques have been used to try to reduce the errors in the linear approximation of a set of Gaussian random variables. In [12], the authors give a detailed treatment of the errors in the MAX_{θ} operation by using error preserving transformations and precomputed lookup tables. These tables are used heuristically to order a set of random variables and compute their statistical maximum. In [17] the authors postpone the computation of the linear maximum during SSTA, if it results in significant non-linearity (distribution skewness is used as a measure of non-linearity). The maximum is propagated as a maximum tuple in such cases. At the primary outputs, a Monte Carlo simulation is performed on the tuple to obtain a better estimation of the circuit delay pdf. The authors in [7] use a moment matching technique to compute non-linear distributions more accurately. Such a technique can be used to get rid of the linearity restriction of the MAX_{θ} operation to reduce the errors in criticality computation.

VI. Reducing Errors

This section describes a simple solution to deal with global errors not eliminated by pruning. We then explore a popular graph reduction technique to speed up criticality computation and finally deal with errors due to independent parameter variations like gate oxide thickness, t_{ox}.

A. LS: Localized Sampling

To tackle edges having global errors (Def. IV.2), we perform a quick localized Monte Carlo sampling of the edges in a cutset Σ, pruned using Algorithm 4. The procedure is described in Algorithm 7. The inputs are Σ; N_{ls} samples of the k independent and identically distributed (i.i.d.) Gaussian principal components (PCs) (Eq. 2) stored in Ψ_p; array R of N_{ls} i.i.d. Gaussian samples for each edge in Σ. Every sample is used to instantiate the edges e_i in Σ (Steps 2-4), from which we compute the edge with maximum delay (Step 5). Array entry $M[i]$ keeps count of the number of samples for which an edge e_i takes on the maximum delay. This helps us compute the global criticality, T_i, of all edges $e_i \in \Sigma$ in Step 7.

Consider a cutset $\Sigma = \{e_1, \ldots, e_n\}$ with edge path delays, $\{e^{\sigma}_1, \ldots, e^{\sigma}_n\}$, represented in terms of the k PCs (for the purpose of simplicity we ignore the spatially uncorrelated random component of variation r_i). In the k-dimensional space, let $\mathcal{R}$ be the region where e^{σ}_i takes on the greatest value in the probability space, i.e., $\mathcal{R}$ is the region of dominance of edge e_i in the cutset. The global criticality T_i of edge e_i is given by the volume integral of the joint pdf of the k i.i.d.

Algorithm 7 LS (Σ, Ψ, R)
// Σ = cutset; $\Psi_p = N_{ls}$ x k array of i.i.d. gaussian samples
// $R = N_{ls}$ x $|\Sigma|$ array of i.i.d. gaussian samples

1: **for** $n = 1$ to N_{ls} **do**
2: **for all** $e_i \in \Sigma$ **do**
3: $d_i = \mu_i + \sum_{j=1}^{j=k} a_{ij} \cdot \Psi_p[n][j] + \zeta_i \cdot R[n][i]$
// $\Psi_p[n][j]$ = value of the j^{th} PC, p_j, and
// $R[n][i]$ = value of random component, r_i,
// for edge e_i at simulation point n (see Eq. 2)
4: **end for**
5: Increment count $M[i]$ of edge e_i with maximum d_i
6: **end for**
7: Compute $T_i = M[i]/N_{ls}$ for all $e_i \in \Sigma$

PCs over $\mathcal{R}$. The LS procedure in Algorithm 7 is a Monte Carlo simulation to compute the volume integral of the joint pdf over region $\mathcal{R}$. The accuracy of LS therefore depends on the number of samples N_{ls} and the accurate computation of the path delay for every edge in the timing graph, or in other words, the forward and reverse SSTA to capture the sensitivities of edge path delays to the k i.i.d. PCs. Intuitively, since edges with high global criticality (large volume integral) have a region of dominance $\mathcal{R}$ near (or including) the origin, the number of samples needed for convergence is not very large. This will be seen in the results Section VII.

It should be noted that we apply the LS procedure to every cutset of the timing graph. The speedup in LS stems from the reduction of the cutset size using the clustering based pruning procedure of Algorithm 4.

B. Timing Graph Reduction

Since we perform a localized sampling on all the levels of the timing graph, G, reducing the number of levels, L, can speed up the runtime. We exploit the fact that the criticality of a node in G is equivalent to the sum of its fanin edge or fanout edge criticalities. To do this, we perform a timing graph reduction (TGR) procedure on nodes with a single fanin or fanout. A straightforward and practical example of this reduction is an inverter chain, wherein a path enters the chain if and only if it passes through all the edges in the chain. Therefore, the criticality of all these edges is the same.

The idea of TGR is borrowed from [14], wherein the objective is to eliminate timing graph nodes to reduce the number of variables and constraints in circuit timing optimization. To perform a TGR we scan timing graph G in the forward and reverse directions merging fanins of single fanout nodes into their fanout and fanouts of single fanin nodes into their fanin respectively. Table II shows the effect of TGR on the number of levels, L, and maximum cutset size, η, on the five largest benchmark circuits. Column 2 shows the size of the circuit. As their names imply, columns "TGR" and "No TGR" are results with and without TGR respectively, applied to G.

C. Spatially Uncorrelated Independent Parameter Variations

Revisiting Eq. 2, independent (spatially uncorrelated) parameter variations like the variation in oxide thickness, t_{ox}, of a transistor, are captured by the single random variable r_i. This is done to avoid tracking the individual contribution of t_{ox} for every transistor in the design as a separate term in the canonical form, as done in [8]. However, errors can occur in the path delay of reconvergent paths, as shown in Fig. 7 taken from one of the ISCAS89 benchmarks.

TABLE II
THE ISCAS89 BENCHMARKS WITH NUMBER OF GATES, N_G, AND INDEPENDENT SOURCES OF VARIATION, N_ζ. THE EFFECT OF TGR ON CIRCUIT DEPTH, L, AND MAXIMUM CUTSET SIZE, η IS ALSO SHOWN.

Name	# of Gates	N_ζ	L		η	
			TGR	No TGR	TGR	No TGR
s13207	7951	22330	63	16	1329	2599
s15850	9772	27290	86	21	1688	2411
s38417	22179	64056	51	13	2821	6638
s35932	16065	56538	33	10	5473	10742
s38584	19253	65512	60	19	5680	10374

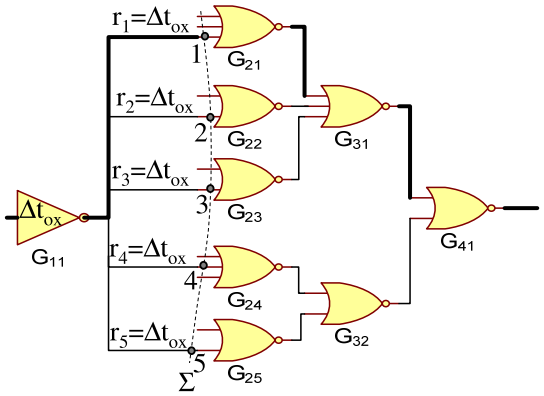


Fig. 7. A reconvergent structure from one of the ISCAS89 benchmarks with a high criticality path indicated using bold lines. Arrival time correlations of fanouts in cutset Σ, denoted r_i, due to variation in oxide thickness, Δt_{ox}, of gate G_{11}, cause structural correlations in reconvergent fanouts like G_{41}.

The figure shows gate G_{11} driving 5 other gates. The arrival time (Def. II.3) at the fanouts of G_{11} consists of a structurally correlated term to capture the variation in the oxide thickness of transistor G_{11}, denoted Δt_{ox}. Since the canonical form consists of a single term to capture spatially uncorrelated variations, r_i, in cutset Σ, these are considered independent, and may cause errors in high criticality paths (shown in bold) particularly when such fanouts have a high degree of correlation. In our experiments, ignoring the structural correlations led to errors of upto 60%, the main culprits being cutsets with reconvergences similar to Fig. 7. Also, to calculate the statistical MAX_σ at the convergence of the paths containing gates $G_{21}, G_{22}, \ldots, G_{25}$, i.e., at gate G_{41}, we need to factor in the common Δt_{ox} of gate G_{11} to reduce inaccuracies in MAX_σ. To keep track of the structural correlations due to spatially uncorrelated independent parameter variations like Δt_{ox}, on encountering a multiple fanout gate like G_{11}, we expand the canonical form of the path delay with its Δt_{ox} variation to accurately compute the arrival time of the downstream

TABLE III
CRITICALITY RUN-TIMES AND ERRORS FOR VARIOUS BENCHMARKS; $\varepsilon = 5\%$ AND $N_{ls} = 1000$ ZSC - ZONE BASED CRITICALITY, ${}^nC_2$ - PAIRWISE PRUNING SCHEME, CPSC - CLUSTERING BASED PRUNING SCHEME, TGR - TIMING GRAPH REDUCTION, LS - LOCALIZED SAMPLING

Metric	Pruning Scheme	Benchmark											
		s3271	s3330	s3384	s4863	s5378	s6669	s9234	s13207	s15850	s38417	s35932	s38584
maximum $\% \ \delta$	ZSC	**44.51**	**36.19**	**43.23**	**31.82**	**59.95**	**40.24**	**38.17**	**41.75**	**44.56**	**34.78**	**21.11**	**48.21**
	${}^nC_2$	4.70	3.82	0.03	17.40	26.94	26.74	36.63	15.72	32.20	30.29	14.95	20.28
	CPSC	4.70	1.42	0.03	17.40	37.41	30.25	36.57	15.64	37.32	30.29	14.95	21.18
	CPSC+TGR+LS	**4.70**	**1.62**	**0.03**	**9.08**	**7.18**	**2.90**	**3.28**	**2.33**	**4.52**	**4.70**	**1.82**	**15.88**
runtime (sec)	ZSC	**0.05**	**0.04**	**0.07**	**0.11**	**0.12**	**0.16**	**0.19**	**0.24**	**0.28**	**1.43**	**1.32**	**1.47**
	${}^nC_2$	0.11	0.09	0.18	0.52	0.51	0.91	1.15	2.36	2.65	58.15	74.69	59.23
	CPSC	0.01	0.01	0.01	0.03	0.02	0.06	0.05	0.03	0.05	0.16	0.36	0.11
	CPSC+TGR+LS	**0.01**	**0.02**	**0.01**	**0.12**	**0.04**	**0.25**	**0.15**	**0.06**	**0.14**	**0.25**	**0.25**	**0.22**
η	ZSC	**622**	**451**	**603**	**528**	**593**	**965**	**644**	**1329**	**1688**	**2821**	**7340**	**5680**
	${}^nC_2$	2	13	1	6	7	11	19	7	8	14	66	15
	CPSC	2	13	1	6	7	11	19	7	8	14	66	15
	CPSC+TGR+LS	**2**	**13**	**1**	**7**	**6**	**12**	**17**	**7**	**8**	**12**	**66**	**15**

gates in the circuit. A similar expansion is performed for gates with multiple fanins while reverse traversing the timing graph to compute the required times of upstream edges. Although the number of terms in the canonical form increases, using a linear sparse array, we only keep track of terms with non-zero sensitivities in the edge path delay. Table II shows the total number of independent sources of variation for the benchmarks under column three, labeled N_ζ. As seen in Section VII, this does not adversely impact the runtime.

VII. RESULTS

Our algorithms were implemented in C++ on top of an SSTA engine [1] and exercised on the 12 largest ISCAS89 benchmarks, with parameter values corresponding to the 100nm technology node [11]. Experiments were conducted on a Linux PC with a 3.0-GHz CPU and 2GB RAM. The average ratio of the standard deviation to the mean of circuit delay was about 12%. We compared four schemes with Monte Carlo simulations using 10000 samples, shown in Table III.

The first scheme is the zone-based ZSC approach in Algorithm 3. Scheme ${}^nC_2$ additionally implements the pairwise pruning strategy of Section V-A with a pruning threshold, $\varepsilon = 5\%$. CPSC implements Algorithm 6 using our clustered pruning and ordering technique. CPSC+LS+TGR performs clustered pruning on the reduced timing graph (TGR) and computes criticalities using the LS procedure (Algorithm 7) with $N_{ls} = 1000$ samples. All approaches excluding ZSC account for structural correlations due to independent parameter variations as described in Section VI-C. Row "maximum $\% \ \delta$" reports the maximum difference between the edge criticality computed using any of the above mentioned schemes and the Monte Carlo simulations, row "runtime" reports the running time in seconds and "η" reports the maximum number of edges in any cutset of the timing graph after pruning. We exclude the times for SSTA and generating the N_{ls} samples in LS.

From Table III, ZSC, which computes criticalities using Clark's MAX_θ formulation results in large errors (the largest being about 60%). As described in Section IV, this is mainly due to the propagation of local errors. CPSC with cutset pruning and ordering does better than ZSC in accuracy and runtime. For circuits exhibiting large global errors, the LS procedure helps reduce them further. Rows in bold compare ZSC with CPSC+TGR+LS. The combined approach greatly reduces the errors and runtime, due to pruning. Moreover, runtime increase is negligible compared to CPSC (an anomaly is s35932 wherein runtime decreases due to TGR). For the 3 large benchmarks we obtain about an order of magnitude difference in run-times of ZSC and the combined approach. Most circuits have errors below 10%, except for s38584. On investigation, it was found that for large fanout structures, path delays themselves (computed in terms of the PCs) contained large errors and hence the LS procedure does not completely eliminate global errors. In terms of the efficacy of our pruning strategy, as expected we vastly outperform the ${}^nC_2$ procedure in runtime (about two orders of magnitude for the larger benchmarks). Each circuit also contained an identical number of edges remaining in the cutsets using the ${}^nC_2$ and CPSC pruning strategies, seen from the entries in row "η".

To evaluate the runtime effectiveness of the zone-based

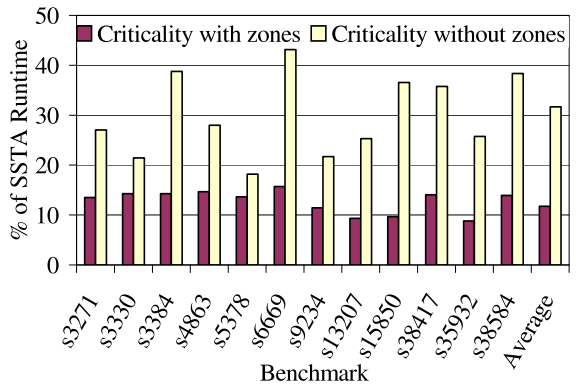


Fig. 8. Runtime of criticality computation as a fraction of SSTA runtime. The two cases shown are with and without the zone-based algorithm to compute the statistical maximum of mc-edges crossing a level in the timing graph.

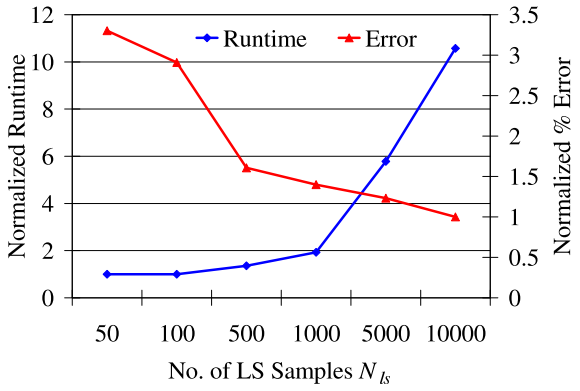


Fig. 9. Tradeoff showing number of LS samples, N_{ls}, vs the overall criticality computation runtime and maximum percentage error (with respect to a Monte Carlo simulation with 10000 samples), for the s38417 ISCAS89 benchmark. Runtimes are normalized to the case with $N_{ls} = 50$ and the error is normalized to the case with $N_{ls} = 10000$.

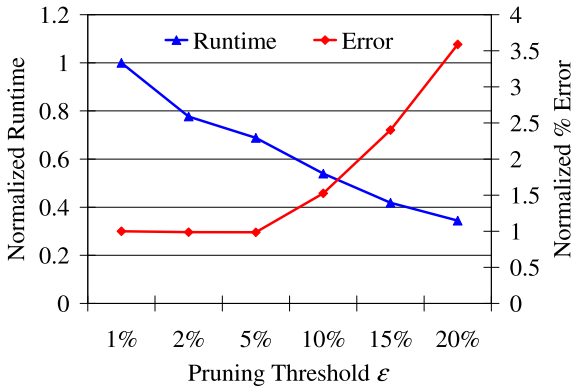


Fig. 10. Tradeoff showing pruning threshold, ε, vs the overall criticality computation runtime and maximum percentage error (with respect to a Monte Carlo simulation with 10000 samples), averaged over the 7 largest ISCAS89 benchmarks. The runtimes and error are normalized to the case with $\varepsilon = 1\%$. The number of samples used in LS, $N_{ls} = 1000$.

approach, Fig. 8 shows the criticality computation runtime with (denoted 'Criticality with zones') and without (denoted 'Criticality without zones') zones as a fraction of the SSTA runtime. In all cases, structural correlations due to independent parameter variations were not taken into account. On average, criticality computation with zones is about $10X$ faster than SSTA and we obtain a speedup of about $2.7X$ in the runtime compared to the case without zones. The runtime for the zone computation procedure of Algorithm 2 on average was less than 0.5% of the SSTA runtime.

In deciding N_{ls}, we observed that as the number of samples increases, the improvement in accuracy diminishes. Fig. 9 shows the tradeoff between the number of samples N_{ls} and the maximum percentage error δ, obtained between our clustering based approach and a Monte Carlo analysis with 10000 runs. As expected, with a small number of LS samples, $N_{ls} < 500$, the error is more than double that with $N_{ls} = 10000$. However, as the number of samples increases, say from 1000 to 5000, the overall runtime almost triples, without much reduction in error. Moreover, as N_{ls} increases, the overall runtime is dominated by the time for LS. In our algorithm, to maintain a reasonable tradeoff of accuracy and runtime, we chose $N_{ls} = 1000$.

Fig. 10 shows the variation of runtime and accuracy (averaged over all benchmarks) when pruning threshold, ε, is varied. With an increase in ε, the cutset size decreases, reducing the overall criticality runtime (mainly due to reduction in the runtime for LS). For pruning thresholds below 5%, the error is relatively constant since the non-dominant edges eliminated do not adversely affect the global criticality of dominant edges. Therefore in our algorithm, we chose a pruning threshold of 5% to obtain good accuracy with a reasonable runtime.

Finally, we implemented the approach of Xiong *et al.* in [15] and compared its performance with our clustering based approach for the benchmarks shown in Table III. For a fair comparison, we ignored independent parameter variations when comparing the two approaches. On average, we obtain a speedup of about $5X$ over the approach in [15]. This is mainly attributed to cutset pruning, which eliminates a large number of non-dominant edges, thereby reducing the number of criticality computations. The advantage of cutset pruning is particularly pronounced for the larger sized benchmarks.

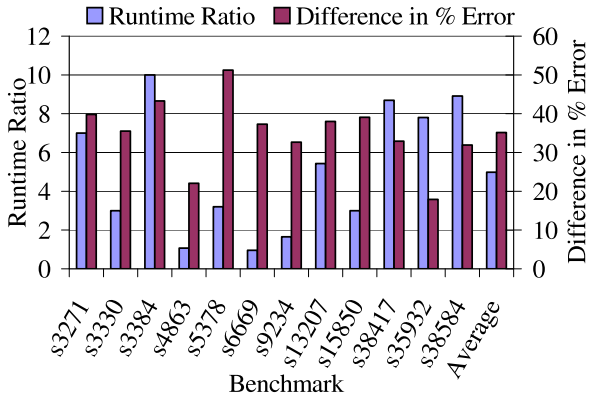


Fig. 11. Comparison of runtime ratio and difference in maximum criticality percentage error between our implementation of the approach in [15] and the clustering based approach, referenced to a Monte Carlo simulation of 10000 samples. The number of samples used in LS, $N_{ls} = 1000$, and the pruning threshold, $\varepsilon = 5\%$.

The difference in maximum percentage error ($\%\ \delta$) when compared to a Monte Carlo simulation of 10000 runs, is shown in Fig. 11 on the secondary axis. On average, over all the benchmarks, we see that if our algorithm reported the maximum criticality difference with a Monte Carlo simulation of $x\%$, the approach in [15] reported a maximum criticality difference of $x + 35\%$. The errors are of similar magnitude to our zone-based scheme, ZSC, implemented without cutset pruning (Table III), since fundamentally both the approaches are similar. Hence, as was seen in the *abc* problem, local and global errors contribute to large overall errors in criticality computation (Table I).

VIII. Conclusion

This paper presents a new linear time technique to compute statistical criticalities in a timing graph. We use the idea of interval zones to process edges crossing multiple cutsets in linear time. We have also developed a new clustering based heuristic capable of both pruning and ordering edges in a cutset to reduce local and global errors resulting from Clark's tightness probability formulation. Our clustering based pruning competes very well with a pairwise pruning strategy with large speedups in runtime. Using our pruning technique with localized sampling and timing graph reduction, our computations produce errors of around 5% when compared to Monte Carlo simulations, even in the face of large gate delay variations. An important topic for future work is to use our clustering based framework to compute criticality incrementally.

References

[1] H. Chang and S. S. Sapatnekar, "Statistical timing analysis under spatial correlations," *IEEE TCAD*, vol. 24, no. 9, pp. 1467–1482, Sep. 2005.

[2] K. Chopra, S. Shah, A. Srivastava, D. Blaauw, and D. Sylvester, "Parametric yield maximization using gate sizing based on efficient statistical power and delay gradient computation," in *IEEE/ACM ICCAD*. IEEE Computer Society, 2005, pp. 1023–1028.

[3] C. E. Clark, "The greatest of a finite set of random variables," *Operations Research*, vol. 9, no. 2, pp. 145–162, Mar-Apr 1961.

[4] M. C. Golumbic, *Algorithmic Graph Theory and Pefect Graphs*. Boston, MA: Elsevier, 2004.

[5] T. F. Gonzalez, "Clustering to minimize the maximum intercluster distance," *Theoretical Computer Science*, vol. 38, no. 2-3, pp. 293–306, 1985.

[6] U. I. Gupta, D. T. Lee, and J. Y.-T. Leung, "Efficient algorithms for interval graphs and circular-arc graphs," *Networks*, vol. 12, no. 4, pp. 459–467, 1982.

[7] X. Li, J. Le, P. Gopalakrishnan, and L. T. Pileggi, "Asymptotic probability extraction for nonnormal performance distributions," *IEEE TCAD*, vol. 26, no. 1, pp. 16–37, Jan. 2007.

[8] X. Li, J. Le, M. Celik, and L. T. Pileggi, "Defining statistical timing sensitivity for logic circuits with large-scale process and environmental variations," *IEEE TCAD*, vol. 27, no. 6, pp. 1041–1054, Jun. 2008.

[9] H. D. Mogal, H. Qian, S. S. Sapatnekar, and K. Bazargan, "Clustering based pruning for statistical criticality computation under process variations," in *IEEE/ACM ICCAD*. IEEE Press, 2007, pp. 340–343.

[10] S. R. Nassif, "Design for variability in DSM technologies," in *IEEE ISQED*. IEEE Computer Society, 2000, p. 451.

[11] Predictive technology model (PTM). [Online]. Available: http://www.eas.asu.edu/~ptm/

[12] D. Sinha, H. Zhou, and N. V. Shenoy, "Advances in computation of the maximum of a set of gaussian random variables," *IEEE TCAD*, vol. 26, no. 8, pp. 1522–1533, Aug. 2007.

[13] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, S. Narayan, D. K. Beece, J. Piaget, N. Venkateswaran, and J. G. Hemmett, "First-order incremental block-based statistical timing analysis," *IEEE TCAD*, vol. 25, no. 10, pp. 2170–2180, Oct. 2006.

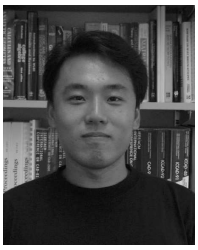
[14] C. Visweswariah and A. R. Conn, "Formulation of static circuit optimization with reduced size, degeneracy and redundancy by timing graph manipulation," in *IEEE/ACM ICCAD*. IEEE Press, 1999, pp. 244–252.

[15] J. Xiong, V. Zolotov, N. Venkateswaran, and C. Visweswariah, "Criticality computation in parameterized statistical timing," in *IEEE/ACM DAC*. ACM Press, 2006, pp. 63–68.

[16] T. Yoshimura and E. S. Kuh, "Efficient algorithms for channel routing," *IEEE TCAD*, vol. 1, no. 1, pp. 25–35, Jan. 1982.

[17] L. Zhang, W. Chen, Y. Hu, and C. Chen, "Statistical static timing analysis with conditional linear max/min approximation and extended canonical timing model," *IEEE TCAD*, vol. 26, no. 8, pp. 1522–1533, Aug. 2007.

Hushrav Mogal received the B.E. degree from the University of Mumbai, Mumbai, India, in Electronics Engineering in 2001 and the M.S. degree in Electrical Engineering from the University of Minnesota at Twin Cities in 2003. He is currently a doctoral candidate at the University of Minnesota Twin Cities, pursuing his Ph.D. in Electrical Engineering. His research interests are in timing analysis and thermal aware CAD.



Haifeng Qian received the B.E. degree from Tsinghua University, Beijing, China, in 2000, the M.S. degree from the University of Texas at Dallas in 2002, and the Ph.D. degree from the University of Minnesota in 2006, all in electrical engineering. Since 2006, he has been a research staff member at the IBM T. J. Watson research center, Yorktown Heights, NY. He received a Best Paper Award at the Design Automation Conference (DAC) 2003, and the ACM Outstanding Ph.D. Dissertation Award in Electronic Design Automation in 2007.

Sachin S. Sapatnekar received the Ph.D. degree from the University of Illinois at Urbana-Champaign in 1992, and is currently on the faculty of the Department of Electrical and Computer Engineering at the University of Minnesota. He has authored several books and papers in the areas of timing and layout, and has held positions on the editorial board of the IEEE Transactions on CAD, the IEEE Transactions on Circuits and Systems II, IEEE Design and Test, and the IEEE Transactions on VLSI Systems. He has served on the Technical Program Committee for various conferences, and as Technical Program and General Chair for Tau and ISPD, and is currently Vice-Chair for DAC. He is a recipient of the NSF Career Award, four conference best paper awards, and the SRC Technical Excellence award.

Kia Bazargan received his Bachelors degree in Computer Science from Sharif University in Tehran, Iran, and his M.S. and PhD in Electrical and Computer Engineering from Northwestern University in Evanston, IL in 1998 and 2000 respectively. He is currently an Associate Professor in the Electrical and Computer Engineering at the University of Minnesota. He has served on the technical program committee of a number of IEEE/ACM sponsored conferences (e.g., FPGA, FPL, DAC, ICCAD, ICCAD, ASPDAC). He was a guest co-editor of ACM Transactions on Embedded Computing Systems (ACM TECS), Special Issue on Dynamically Adaptable Embedded Systems in 2003. He is an Associate Editor of IEEE Transaction on Computer- Aided Design of Integrated Circuits and Systems. He was a recipient of NSF CAREER award in 2004.