Short Papers

Entropy-Driven Parity-Tree Selection for Low-Overhead Concurrent Error Detection in Finite State Machines

Sobeeh Almukhaizim, Petros Drineas, and Yiorgos Makris

***Abstract*—This paper presents discuss the problem of parity-tree selection for performing concurrent error detection (CED) with low overhead in finite state machines (FSMs). We first develop a nonintrusive CED method based on compaction of the state/output bits of an FSM via parity trees and comparison to the correct responses, which are generated through additional on-chip parity prediction hardware. Similar to off-line test-response-compaction practices, this method minimizes the number of parity trees required for performing lossless compaction. However, while a few parity trees are typically sufficient, the area and the power consumption of the corresponding parity predictor is not always in proportion with the number of implemented functions. Therefore, parity-tree-selection methods that minimize the overhead of the parity predictor, rather than the number of parity trees, are required. Towards this end, we then extend our method into a systematic search that exploits the correlation between the area and the power consumption of a function and its entropy, in order to select parity trees that minimize the incurred overhead. Experimental results on benchmark circuits demonstrate that this solution achieves significant reduction in area and power consumption over the basic method that simply minimizes the number of parity trees.**

***Index Terms*—Concurrent error detection (CED), entropy, on-line test, parity trees.**

I. Introduction

Parity constitutes a powerful tool for examining the functional correctness of a circuit and has been extensively used in both off-line- [1]–[3] and on-line-test [4]–[6] methods. In an off-line test, for example, parity trees are often employed to perform lossless compaction of the circuit responses, in order to reduce the amount of test data. Since the correct responses are stored in the memory of the automatic test equipment (ATE), the optimization objective of this compaction is the minimization of the number of parity trees necessary and, by extension, the corresponding ATE bandwidth and storage requirements. The same compaction principle can also be used for performing on-line concurrent error detection (CED). In this case, however, the correct responses are not explicitly stored but are generated through additional parity prediction functions implemented in hardware. Therefore, the compaction optimization objective should be the minimization of the overhead incurred by the parity prediction hardware, rather than the minimization of the number of parity trees. Interestingly, selecting the minimum number of parity trees does not necessarily result in a minimal overhead implementation of the parity predictor.

Manuscript received August 5, 2004; revised April 11, 2005 and June 21, 2005. This paper was recommended by Associate Editor K. Chakrabarty.

S. Almukhaizim is with the Department of Electrical Engineering, Yale University, New Haven, CT 06520-8285 USA (e-mail: sobeeh.almukhaizim@yale.edu).

P. Drineas is with the Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180-3522 USA (e-mail: drinep@cs.rpi.edu).

Y. Makris is with the Department of Electrical Engineering and Computer Science, Yale University, New Haven, CT 06520-8285 USA (e-mail: yiorgos.makris@yale.edu).

Digital Object Identifier 10.1109/TCAD.2005.855933

In this paper, we develop a low-overhead solution to the problem of nonintrusive parity-based CED in finite state machines (FSMs). First, in Section III, we demonstrate the general principle of this CED method, which is based on compaction of the state/output bits of an FSM via parity trees and comparison to the correct responses that are generated by on-chip parity prediction functions. We follow the paradigm of off-line-test methods and we set our optimization objective as the minimization of the number of parity trees necessary for lossless compaction. We formulate the problem as an integer program and we employ an algorithm based on linear-programming relaxation and randomized rounding to identify feasible solutions. The compaction ratio achieved by this method is typically substantial. This implies that the number of parity prediction functions implemented in hardware is significantly reduced as compared to duplication, which is the extreme case of nonintrusive parity-based CED with zero compaction ratio. Nevertheless, as explained through an example in Section IV, the corresponding savings in area and power are not always commensurate with this reduction.

To address this problem, in Section V, we then propose a systematic method for selecting parity trees based on the overhead incurred by the corresponding parity prediction functions. The proposed method estimates the overhead of potential solutions by estimating the entropy of the parity prediction functions. Cheng and Agrawal [7] pioneered the use of entropy for estimating the complexity of a multioutput function. Their observations have been followed by other works [8]–[11], establishing the correlation between the entropy of a function and the area and power-consumption overhead incurred by its implementation. Hardware estimators that are more accurate than entropy would certainly be of great interest, yet to the best of our knowledge, no such estimators have been proposed either in the theoretical computer science or the CAD community. The proposed algorithm is based on the following three principles:

1) Low entropy corresponds to nonrandom functions and thus, to cheaper and less power-consuming hardware implementations.
2) Entropy estimation is feasible via uniform sampling.
3) A biased random walk in the set of feasible solutions, using entropy as the potential function and a bias towards entropy minimization, converges to optimal feasible solutions.

We emphasize the high complexity of the problem at hand, which arises from the twofold objective of achieving lossless compaction while minimizing the overhead incurred by the parity predictor. Although lossless compaction can be expressed as a linear optimization problem, overhead minimization leads to nonlinearities that are much harder to handle. In particular, these nonlinearities arise from the attempt to minimize and compute the entropy of candidate predictor circuits. In order to address these issues, we combine several algorithmic techniques. Randomized rounding is again used to solve the integer-linear-programming (ILP) formulation of lossless compaction; Monte Carlo sampling techniques are used to estimate the entropy of a function and, by extension, the incurred overhead; and the Markov-chain–Monte Carlo method and, in particular, the metropolis random walk are used to minimize an entropy-based potential function.

As demonstrated through experimental results on benchmark FSMs in Section VI, the basic method that selects the minimal number of parity trees yields a CED solution that requires on average 52.04% fewer parity prediction functions than duplication, yet only saves an average

of 20.72% in area and 20.97% in power consumption. However, when parity-trees are selected by the entropy-driven algorithm, additional savings of 24.81% in area and 23.16% in power consumption are achieved.

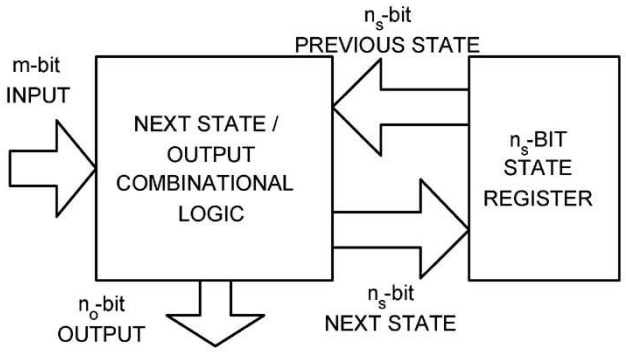
II. Related Work in Cost-Driven Parity-Tree Selection

The only previous work wherein parity-tree selection is driven by the actual hardware cost of the implementation is [5]. This method partitions the outputs of the circuit into multiple groups and computes the parity of each of these groups using parity trees. A check symbol generator generates the expected parity of each of these groups and a checker compares the computed parity to the expected parity. The method proposed in [5] ensures that any single fault will affect at most one output of each group by resynthesizing the circuit such that no logic sharing exists between outputs belonging to the same group. To minimize the incurred area overhead, a cost function is used, which reflects the exact total cost of the resynthesized circuit and the CED overhead as obtained through actual synthesis. Subsequently, they guide the heuristic formation of the parity groups towards minimization of this cost function.

The method in [5] shares many similarities with our method. Both methods detect any error that results in an odd number of incorrect output bits in one or more groups and use a simple comparator as the checker. However, the two methods have many fundamental differences. First, the method in [5] is intrusive, since it resynthesizes the circuit, while the proposed method is nonintrusive, since the original circuit is intact. Second, the cost function of [5] is exact, based on synthesis of potential solutions. In contrast, our method uses an entropic potential function that allows rapid exploration of a larger search space. Finally, the method in [5] employs structural analysis to identify the set of errors to be addressed. In addition to this structural analysis, which our method also uses for large circuits, we provide the option of functional analysis via fault-simulation to identify a more accurate set of errors of interest.

III. Parity-Based CED for FSMs

In this section, we develop a parity-based implementation of the general algebraic CED method for FSMs [12], wherein a set of parity trees performs lossless compaction of the circuit responses. Additional hardware is subsequently used to predict the compacted error-free responses, and a comparator is employed to identify any discrepancy between the output of the compactor and the output of the predictor. In an effort to reduce the incurred overhead, this method minimizes the number of parity trees required for lossless compaction and, by extension, the number of parity prediction functions that will be implemented on-chip. The proposed method detects all errors in a specified error model. Such a model is prescribed by providing the error-free response and all erroneous responses of interest for every FSM transition. We note that this general method can accommodate any error model, regardless of whether they result from a single or multiple error sources. Target error models are expected to be restricted, in the sense that the set of resulting erroneous responses should be a subset of all possible circuit responses. Indeed, for an unrestricted error model, wherein an error-free response may be transformed into any erroneous response, information theory proves that any nonintrusive CED circuit will be as complex as the original circuit [13], making duplication the most appropriate solution. However, more cost-effective solutions may be devised through compaction [14] when a restricted error model is specified.



(a)

ERROR DETECTABILITY TABLE

	b_1	b_2	...	b_{ns}	b_{ns+1}	b_{ns+2}	...	b_{ns+no}
Erroneous $Case_1$	1			1				1
Erroneous $Case_2$		1						
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
Erroneous $Case_f$		1					1	1

(b)

Fig. 1. FSM example and EDT.

A. Method Overview

Consider the next state/output combinational logic of the FSM shown in Fig. 1(a), which has m inputs and $n = n_s + n_o$ outputs, out of which n_s are state bits and n_o are outputs bits. For every combination of input and previous state, any error will manifest itself as a difference between the correct response and the erroneous response. This difference is detectable in a nonempty set of state and output bits. Each such set constitutes an erroneous case (EC). Clearly, several combinations of a transition and an error may lead to the same EC, i.e., the same set of bits on which the error may be detected during the transition. The set of all ECs may be represented in the tabular format of Fig. 1(b), where columns correspond to the n state/output bits, rows correspond to the f distinct ECs, and entries of "1" in the table indicate the state/output bits at which each EC is detectable [15]. This error detectability table (EDT) is the mechanism for prescribing any target error model.

Given a model for the source of the errors (e.g., single-line faults), construction of the EDT can be done in two ways, depending on the size of the circuit. For small circuits, exhaustive fault simulation can be performed to construct an EDT that contains the exact ECs that can occur. For large circuits where exhaustive fault simulation is not an option, a structural analysis based on cones of logic, similar to the one performed in [5], can be employed. Since the functionality of the circuit is not taken into account in this case, the resulting EDT is pessimistic, in the sense that it contains ECs that may never occur in practice. While this places additional unnecessary constraints that may lead to a more expensive solution, nevertheless, it enables the application of the proposed method to small and large circuits alike.

Detecting all errors requires that at least one state/output bit in each EC is predicted through additional hardware and compared to its actual run-time value. Instead of duplicating the circuit, however, we employ state/output compaction via parity trees. The key observation is that the parity (XOR) function of several state/output bits, an odd number of which detects an EC, also detects the EC. Therefore, it is possible that a small number of parity functions compacting the state/output bits

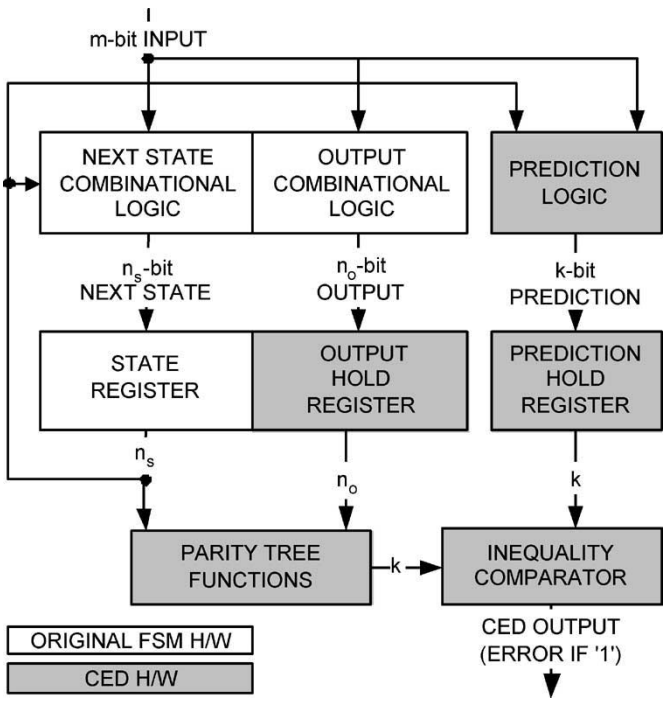


Fig. 2. Parity-based CED method.

will be adequate to cover all ECs in a prescribed error model. Using the information in the EDT, the optimization objective of our method is to minimize the number of parity bits k that need to be constructed out of the next state/output bits such that all ECs are detected. An EC is detected by a parity tree if and only if the parity tree comprises an odd number of bits that detect the EC.

Based on the above observations, the proposed methodology is rather straightforward, as depicted in the form of a block diagram in Fig. 2. Given an FSM, k parity trees are used for lossless compaction of the state/output bits. Combinational logic is employed to predict the values of the k bits that compact the n state/output bits for each FSM transition, and a comparator is used to detect any discrepancy. Similar to [16], registers are added to hold the output and the predicted parity so that comparison is performed one clock cycle later to detect faults in the state register. Thus, all FSM errors are detected without latency but are reported one clock cycle later.

B. Lossless Compaction via ILP

We now demonstrate how to model parity-tree selection for lossless compaction as a set of integral inequalities. First, in Section III-B1, we review the necessary notation. Then, in Section III-B2, we formulate a set of integral constraints that detect all ECs, given a fixed number of parity trees. Subsequently, in Section III-B3, we discuss the use of randomized rounding for identifying feasible points, namely points that satisfy all the constraints of the integer program. Then, in Section III-B4, we present the overall algorithm that performs binary search to minimize the number of necessary parity trees.

1) Notation: We start by introducing some notations and useful facts that will be repeatedly used throughout this section. Let $[x]$ denote the sequence $1, 2, 3, \ldots, x$ for any nonzero positive integer x. Given an FSM, let m be the total number of inputs of the FSM, and let $n = n_s + n_o$ be the total number of outputs of the next state/output combinational logic, which we denote by $\{b_1, b_2, \ldots, b_n\}$. The set of ECs that need to be detected is denoted by $\mathcal{F} = \{\text{EC}_1, \text{EC}_2, \ldots, \text{EC}_f\}$, where $|\mathcal{F}| = f$. The EDT of Fig. 1(b) is stored in a matrix, which we denote by V. The dimensions of matrix V are $f \times n$, and we denote its (i, j)th element by $V(i, j)$ for all $i \in [f]$, $j \in [n]$. We remind the reader that for Boolean variables x_1, x_2, $x_1 \oplus x_2 = (x_1 + x_2) \bmod 2$. Also, any subset of $\{b_1, b_2, \ldots, b_n\}$ may be represented by an n-dimensional 0–1 vector, e.g., the subset $\{b_1, b_3, b_4\}$ may be represented by $[1\ 0\ 1\ 1 \ldots 0]$.

2) Integer Program Formulation: We first define the entries of the $f \times n$ matrix V, which can be either 0 or 1.

Definition 1: For all $i \in [f]$, $j \in [n]$; $V(i, j)$ is set to 1 if and only if EC_i is detectable by the jth output bit b_j. Otherwise, $V(i, j)$ is set to 0.

Our problem may now be stated as follows.

Statement 1: Given a positive integer k, find k subsets $\beta_1, \ldots, \beta_k$ of $\{b_1, b_2, \ldots, b_n\}$ such that

$$\mathbf{cov}(\oplus\beta_1) \cup \mathbf{cov}(\oplus\beta_2) \cup \cdots \cup \mathbf{cov}(\oplus\beta_k) = \mathcal{F}$$

or report the lack thereof.

Here, $\oplus\beta_\ell$ (for all $\ell \in [k]$) denotes the parity tree formed by the next state/output bits in β_ℓ and $\mathbf{cov}(\oplus\beta_\ell)$ denotes the ECs detected by this parity tree. An EC_i is detected by the parity tree formed by the bits in β_ℓ if and only if

$$\left(\sum_{b_y \in \beta_\ell} V(i, y) \right) \bmod 2 \geq 1.$$

The above formula essentially checks whether the XOR of the bits in β_ℓ detects EC_i. Thus, we can check whether the k parity trees (the parity trees corresponding to β_ℓ, for all $\ell \in [k]$) detect all ECs. The problem may now be stated in a matrix notation.

Statement 2: Given a positive integer k, find k n-dimensional 0–1 vectors $\beta^{(1)}, \ldots, \beta^{(k)}$ such that

$$\sum_{\ell=1}^{k} \left[V \cdot \beta^{(\ell)} \bmod 2 \right] \geq \vec{\mathbf{1}}_\mathbf{f}$$

or report the lack thereof. Here, $\vec{\mathbf{1}}_\mathbf{f}$ is an f-dimensional vector of 1's.

We now remove the mod operator by adding new variables.

Statement 3: Given a positive integer k, find vectors $\beta^{(\ell)}$, $r^{(\ell)}$, $w^{(\ell)}$, $\ell \in [k]$, such that

$$\begin{aligned}
V \cdot \beta^{(1)} &= 2 \cdot w^{(1)} + r^{(1)} \\
V \cdot \beta^{(2)} &= 2 \cdot w^{(2)} + r^{(2)} \\
&\vdots \\
V \cdot \beta^{(k)} &= 2 \cdot w^{(k)} + r^{(k)} \\
r^{(1)} + \ldots + r^{(k)} &\geq \vec{\mathbf{1}}_\mathbf{f} \\
\beta^{(\ell)} &\in \{0, 1\}^n \\
r^{(\ell)} &\in \{0, 1\}^f \\
w^{(\ell)} &\in \left\{ 0, 1, \ldots, \left\lfloor \frac{n}{2} \right\rfloor \right\}^f
\end{aligned}$$

or report the lack thereof.

In order to understand the above constraints, observe that the remainder $r^{(1)}$ is an f-dimensional 0–1 vector denoting whether $\text{EC}_1, \ldots, \text{EC}_f$ are detected by the parity tree corresponding to $\beta^{(1)}$. We note that $w^{(1)}$ is also an f-dimensional vector that removes the mod 2 operation. Each element in the sum of the remainders $r^{(\ell)}$ is required to be at least one, in order to guarantee that all ECs are detected.

matrix V, k
vectors $\beta^{(\ell)}$ that detect all ECs
$ITER = 10^3$;
Find a feasible point for the LP of Statement 3;
repeat
  Use randomized rounding to create an integer solution;
until *all constraints of Statement 3 are satisfied* **or** $ITER$ *repetitions are exceeded*;
if *all constraints of Statement 3 are satisfied* **then**
  Report the $\beta^{(\ell)}, \ell \in [k]$;
else
  Fail;
end

Algorithm 1. Randomized-rounding algorithm.

matrix V
vectors $\beta^{(\ell)}$ that detect all ECs
$ITER = 10^3$; $left = 1$; $right = n$;
while $left < right$ **do**
  $k = \lfloor (right - left)/2 \rfloor$;
  if *Algorithm 1 succeeds on input* V, k **then**
    set $right = k$;
  else
    set $left = k$;
  end
end
Report the minimal k and $\beta^{(\ell)}, \ell \in [k]$, such that all the constraints of Statement 3 are satisfied.

Algorithm 2. Parity-tree-count-minimization algorithm.

3) Randomized Rounding: In Statement 3, we described our problem as an integer program. Our goal is to find a feasible point; namely, values for all $r^{(\ell)}$, $w^{(\ell)}$, and $\beta^{(\ell)}$ such that all the constraints of Statement 3 are satisfied. In order to identify a feasible point for the integer program, we employ a technique called randomized rounding [17]. The idea of randomized rounding is simple: Solve the linear program relaxation of the integer program, which is easily done in polynomial time using, for example, the ellipsoid [18] or interior-points methods [19]; and round the resulting real values probabilistically. More specifically, if $\tilde{x}$ denotes such a value, we create x as follows:

$$x = \begin{cases} 1, & \text{with probability } \tilde{x} \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

It is easy to argue that, using this randomized-rounding scheme, the constraints of Statement 3 are satisfied in expectation. However, the variance is quite large; and thus, the constraints might be violated. Raghavan and Thompson [17], who were the first to introduce this simple but powerful randomized-rounding scheme, argue that simple modifications of the rounding probabilities are sufficient to guarantee that all the constraints are satisfied with high probability instead of in expectation. In practice, we probabilistically round the resulting real values for a fixed number of times and explicitly verify that the resulting solution satisfies all the constraints. Algorithm 1 answers the existence problem for a given number of parity trees k.

4) Parity-Tree-Count-Minimization Algorithm: The above formulation answers the existence problem for a given number of parity trees k. However, our objective is to minimize k, which is now straightforward. If we can solve the problem of Statement 3 in time T, then we can easily minimize k in $T \log n$ time: Since $1 \leq k \leq n$, we can perform binary search and find the optimal k, as shown in Algorithm 2.

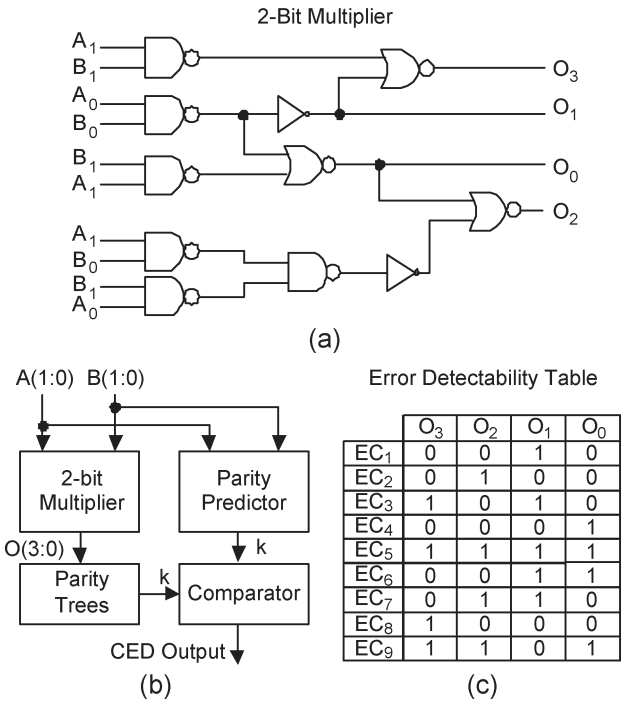


Fig. 3. Motivation example. Parity-based CED on a 2-bit multiplier.

IV. Limitations of Parity-Tree Count Minimization

The optimization objective of the parity-based CED methodology described in the previous section is the minimization of the number of parity trees necessary for lossless compaction. The rationale behind this objective is to minimize the number of parity prediction functions that will need to be built on-chip and, by extension, the corresponding overhead in area and power consumption. However, as we demonstrate through a toy example in this section, this expectation is not always met in practice.

Consider the gate-level implementation of a 2-bit multiplier shown in Fig. 3(a), and assume that we want to perform the proposed parity-based CED as shown in Fig. 3(b). For the purpose of defining a restricted error model for this example, we simulated all single errors in the circuit for all possible input combinations and we obtained the nine ECs shown in the EDT of Fig. 3(c). Essentially, this means that any single error will result in a discrepancy between the correct and erroneous response in one of these nine sets of output bits. The problem now reduces to choosing, out of the 15 possible parity trees that can be constructed from the four outputs, the minimum number k that compact the outputs while preserving the detectability of all ECs. This can be done through the ILP formulation and the solution based on randomized rounding that were described in the previous section. On the multiplier, these methods yield the minimum number of parity trees that detect all ECs, $k = 2$. One such solution is $P_1 = O_1 \oplus O_0$ and $P_2 = O_3 \oplus O_2 \oplus O_1$. When we synthesize the parity predictor for the two parity functions P_1 and P_2, the area overhead is 22 gates, and the power-consumption overhead estimated by SIS [20], based on the switching activity in the circuit, is 178.8 μW. Surprisingly, these two functions are even more expensive than the circuit itself, which generates four functions and incurs an area overhead of 11 gates and a power-consumption overhead of 173.9 μW. Yet other solutions exist that yield a cheaper parity prediction circuit. For example, another set of $k = 2$ parity trees, $P_3 = O_3 \oplus O_1$ and $P_4 = O_3 \oplus O_2 \oplus O_0$, will also detect all ECs and the corresponding parity predictor only incurs an area overhead of ten gates and a power-consumption overhead of

155.6 μW. Moreover, there exists a solution with $k = 3$ parity trees, $P_3 = O_3 \oplus O_1$, $P_5 = O_0$, and $P_6 = O_3 \oplus O_2$, which also detects all ECs and has a parity predictor that incurs an area overhead of only seven gates and a power dissipation overhead of only 138 μW.

As demonstrated through this toy example, the number of on-chip parity prediction functions is not the most accurate indication of the incurred overhead. The underlying reason is that functions implemented by practical circuits are typically structured and simple, resulting in inexpensive implementations. Yet when XORed together they may result in random functions that are expensive to implement. Therefore, minimizing the number of parity trees is not the best optimization objective for methods that generate the responses through on-chip hardware. Rather, methods that select parity trees for lossless compaction based on the actual overhead incurred by the parity predictor are required. Such a method, based on the entropy of the predictor, is proposed in the following section.

V. Entropy-Driven Parity-Tree Selection

We start by briefly sketching our approach. First, given a circuit and its EDT, we use Algorithm 1 to find a specific feasible solution, namely a set of k parity trees for compacting the circuit outputs that detect all ECs. Notice that the output of the k parity trees is a k-bit 0–1 string. Then, we estimate the hardware cost of the predictor associated with this specific set of parity trees by approximating its entropy, as detailed in Section V-A. Again, notice that the predictor is a function whose domain is the set of all possible inputs to the circuit and whose output is a k-bit string. We then examine a neighboring feasible solution, namely a solution that preserves detection of all ECs and is "close" to the previous solution in some well-defined proximity metric. Subsequently, we estimate the overhead of the new solution, again by estimating its entropy. We then accept or reject a move to this neighbor with a probability that depends on the estimated overhead of the new solution. More specifically, "better" solutions are almost always accepted, while "worse" solutions are accepted with a probability that becomes exponentially smaller as their quality decreases. This is essentially the classical Metropolis Algorithm, described in detail in Section V-B. The Metropolis Algorithm is the basis of simulated annealing techniques [21] that have been very successful in tackling nonlinear optimization problems in various contexts. The Overall Algorithm, which repeats the above procedure for all values of k from 1 to n and selects the best solution, is presented in Section V-C.

A. Hardware Estimation via Entropy

We start by defining the entropy of the parity-predictor function. Recall that the domain of this function is $\{0,1\}^m$ (all possible input combinations) and its range is $\{0,1\}^k$, where k is the number of parity trees. Let y denote the output of the predictor, which is a k-bit string of zeros and ones. Then, given a set of parity trees $\mathcal{B} = \{\beta^{(1)}, \ldots, \beta^{(k)}\}$, the entropy of the corresponding parity-predictor function is

$$H(\mathcal{B}) = -\sum_{x \in \{0,1\}^k} \mathbf{Pr}(y = x) \cdot \log_2 \mathbf{Pr}(y = x). \qquad (2)$$

In order to put the above formula in words, we notice that the variable x assumes all possible values in $\{0,1\}^k$. The parity-predictor function (the set of parity trees $\mathcal{B}$) has 2^m possible input combinations and each such combination returns a specific k-bit output string. The $\mathbf{Pr}(y = x)$ counts the number of times that the output of the parity predictor (y) is equal to a fixed string x over all possible 2^m input combinations of the parity predictor divided by 2^m. The entropy ranges between 0 and k. Smaller values imply that the function is less random; and hence, its implementation overhead is expected to be low.

A set of k parity predictor trees $\mathcal{B}$
Approximate entropy H
$\ell = 100k2^{k/4}$; S = {};
for $i = 1 \ldots \ell$ **do**
  /* *Create a sample of size* ℓ */
  Pick a random input $IN \in \{0,1\}^m$;
  Compute the k-bit output y of the parity predictor function given input IN;
  S = S $\cup$ y;
end
foreach $x \in \{0,1\}^k$ **do**
  /* *Compute the frequencies of the elements in the sample* */
  $i =$ decimal value of x;
  $t =$ number of times x appears in $\mathcal{S}$;
  $q_i = t/\ell$;
end

/* *Approximate the entropy of the "large" elements* */
$\mathcal{Q} = \{i : q_i > 1/2^{k/4}\}$;
$H_1 = -\sum_{x \in \mathcal{Q}} q_i \log_2 q_i$;

/* *Approximate the entropy of the "small" elements* */
$\bar{\mathcal{Q}} = \{0, 1, 2, \ldots, 2^k - 1\} - \mathcal{Q}$;
$H_2 = \frac{k}{2} \sum_{i \in \bar{\mathcal{Q}}} q_i$;

/* *Return the approximate entropy* */
$H = H_1 + H_2$;

Algorithm 3. The approximate entrophy algorithm [22].

On the other hand, values of entropy close to k imply that the function is random; and hence, its implementation overhead is expected to be high. Exact computation of the entropy takes time exponential in m. Fortunately, we can approximate the entropy using sampling. The technique that we use was presented by Batu *et al.* in [22], and we briefly summarize it in Algorithm 3. The interested reader might seek more details in [22].

B. Cost-Driven Selection Using the Metropolis Filter

Let Ω denote the set of all possible feasible solutions to the ILP of Statement 3. Each feasible solution is a set of vectors $\mathcal{B} = \{\beta^{(1)}, \beta^{(1)}, \ldots, \beta^{(k)}\}$, and recall that each $\beta^{(i)}$ belongs to $\{0,1\}^n$. Let each element of Ω be a vertex of a graph G. We will perform a biased random walk on G. However, we first need to define the edges of G. An edge in a graph usually denotes some "similarity" between the corresponding vertices. In our case, two vertices (feasible solutions) will be "similar" if at most t elements of their corresponding sets of vectors (parity trees) are different, where $t < k \cdot n$ is a small constant value. Intuitively, this notion of similarity means that the two feasible solutions corresponding to the vertices of G have almost the same parity trees, with at most t changes. Using this "similarity" definition, we can construct the graph G. Notice that the maximal possible number of neighbors of a vertex is exactly equal to $MAX = \sum_{j=1}^{t} \binom{kn}{j}$. Clearly, not all these neighbors correspond to feasible solutions. We can now construct a Markov chain in the form of a biased random walk in G, as shown in Algorithm 4.

This random walk essentially corresponds to the Metropolis random walk [23]. We note here that the constant 2 in (2) could be replaced by any constant $\alpha > 1$. Notice that if $H(\mathcal{B})$ is larger than $H(\tilde{\mathcal{B}})$, namely the solution at w is better than the solution at v, then the move happens with a probability of 1. Otherwise, the probability of moving drops rapidly. Usually, values of α close to 1 correspond to random walks that explore a large fraction of Ω and, thus, are slow. On the other hand, large values of α (e.g., $\alpha \to \infty$) correspond to myopic greedy approaches that are fast, but could get stuck in local optima.

Find a vertex v of G by solving the ILP of Statement 3 using randomized rounding;
$M = 1000$;
while *a move happened in the last M steps* **do**
  / Find the feasible neighbors of v */*
  - Construct all MAX neighbors of v;
  - Call Algorithm 1 and construct $\Gamma(v)$, the set of **feasible** neighbors of v;
  - Let $d(v) = |\Gamma(v)|$ denote the *degree* of v in the graph G;
  / Move to a neighbor of v, or stay */*
  - With probability 1/2 stay at v and **break**;
  - With probability $\frac{1-d(v)}{MAX}$ stay at v and **break**;
  - Select one of the $d(v)$ neighbors of v with probability $\frac{1}{MAX}$. Let w denote this neighbor;
  - Let $\mathcal{B}$ denote the set of parity trees corresponding to the feasible solution of vertex v;
  - Approximate the entropy $H(\mathcal{B})$ using Algorithm 3;
  - Let $\tilde{\mathcal{B}}$ denote the set of parity trees corresponding to the feasible solution of vertex w;
  - Approximate the entropy $H(\tilde{\mathcal{B}})$ using Algorithm 3;
  - Move to w with probability

$$\min\left\{1, 2^{H(\mathcal{B})-H(\tilde{\mathcal{B}})}\right\}. \quad (2)$$

  - Set $v = w$;
end

Algorithm 4. Metropolis-random-walk algorithm.

for $k = 1 \ldots n$ **do**
  Run the Metropolis random walk (Algorithm 4);
  Keep the final solution and divide its entropy by k;
end
Return the solution with the lowest normalized entropy;

Algorithm 5. Overall algorithm.

Finally, the choice of t also impacts our approach. Small values of t (e.g., $t = 1$, $MAX = kn$ neighbors per vertex) result in graphs with a small number of edges and usually many independent connected components. Obviously, a random walk in such a graph is unable to explore the solution space Ω efficiently. In our experiments with benchmark circuits, we observed that small perturbations of feasible solutions resulted to new feasible solutions.

C. Overall Algorithm

The overall entropy-driven-selection method is shown in Algorithm 5. In essence, for every possible value of k, we run the Metropolis random walk on the graph G generated by the set of feasible solutions to the ILP of Statement 3. Recall that k is the number of parity trees and ranges from 1 up to n. We normalize the entropy of the best solution for each k by dividing by k, and we pick the solution with the lowest normalized entropy. Notice that larger values of k may return solutions with lower overhead; thus, we have to run the random walk for all possible values of k.

VI. Experimental Results

In this section, we evaluate experimentally the overhead of the proposed methods for parity-based CED in FSMs. First, in Section VI-A, we describe the setup of the experiments. Then, in Section VI-B, we compare the overhead of parity-based CED with the minimum-tree-count number selection, which was described in Section III, to the overhead of duplication-based CED. Finally, in Section VI-C, we compare the overhead of parity-based CED with entropy-driven tree selection, which was described in Section V, to the overhead of parity-based CED with minimum-tree-count number selection.

A. Experimental Setup

We experiment with the standard set of Microelectronics Center of North Carolina (MCNC) and ITC99 benchmark FSM circuits. The circuits are synthesized using the rugged script of SIS [20] and are mapped to a standard library containing only 2-input gates. For the purpose of comparison, we first implement the duplication-based CED method. Then, we construct the EDT for all single errors. For the MCNC benchmark circuits, we perform exhaustive functional fault simulation using internally developed software, which is built around the fault simulator HOPE [24] and identifies all pairs of error-free and erroneous responses. For the ITC99 benchmark circuits, we perform the structural analysis method of [5], and we include an entry in the EDT for every possible subset of outputs driven by each line in the circuit. Starting from the EDT, we first apply the parity-based CED method that selects the minimum number of trees $k_{\min}$, which was described in Section III and which we implemented around linear programming solver (lpsolve) [25]. Finally, we use internally developed software that implements the entropy-driven parity-tree-selection method of Section V.

We now discuss our choices for the parameters α, t, and M, as defined in Algorithm 4 and Section V-B. We experimented with values of α ranging from 1 to 5, in increments of 0.2, as well as with a few very large values of α (i.e., 10^2, 10^3, 10^4, 10^5). The latter values were poor choices, since our random walk converged immediately to inferior (expensive) solutions. Among the former values, $\alpha = 2$ was typically a good choice, and all our experimental results are reported for $\alpha = 2$. We conjecture that small values of α should typically be the best choice for exploring the state space of the entropy-driven parity-tree-selection problem. Regarding t, we first experimented with $t = 1$, which resulted in a very disconnected graph with many independent components and very expensive solutions to the problem. The $t = 2$ value already gave significant improvements, as reported in our experimental results, whereas larger values of t (i.e., 3 and 4) marginally decreased the overall cost of the solution, at the expense of significantly increasing the running time of the algorithm. Finally, $M = 10^3$ iterations sufficed to detect convergence (in a global or local optimal point) in all benchmark circuits. For a subset of the circuits, we experimented with larger values of M (up to 10^5), without being able to improve the hardware complexity of the predictor.

The key conjecture of this work is that entropy can be used as a cost function to drive the selection of parity trees with a low-area and low-power-consumption parity-predictor implementation. With regards to delay overhead, the parity predictor of the proposed method operates in parallel with the next state/output logic of the FSM, in the same way that a replica of the next state/output logic operates in parallel with the circuit in duplication-based CED. Thus, as long as the parity predictor is faster than the next state/output logic of the FSM, no delay is incurred on the operational clock period of the FSM. While this holds true for all circuits in our experiments, we refrain from emphasizing this point since no correlation between the delay of the parity predictor and its entropy can be inferred.

B. Duplication Versus Minimum Parity-Tree Count

The results of duplication-based CED and parity-based CED with minimum tree count are summarized in Table I. Under the first and second major headings, we provide details about the FSM circuits that were used: name, number of primary inputs, number of state bits, and

TABLE I
DUPLICATION-BASED CED VERSUS PARITY-BASED CED WITH MINIMUM TREE COUNT

Benchmark	Circuit Details			Duplication-Based CED				Parity-Based CED with Minimum Tree Count			
Circuit Name	Input Bits	State Bits	Output Bits	Parity Functions	Number of Gates	Area Cost	Power Consumption	Parity Functions	Number of Gates	Area Cost	Power Consumption
cse	7	4	7	11	196	256128	528.7	5	131	171680	454.1
donfile	2	5	1	6	97	128064	408.8	4	57	74704	247.9
dk16	2	5	3	8	240	316448	983.8	6	323	428736	1182.2
dk512	1	4	3	7	74	96048	308.5	4	79	104400	340
keyb	7	5	2	7	228	298352	742.7	5	82	107648	325.3
pma	8	5	8	13	347	453792	888.8	6	186	243136	644.2
sse	7	4	7	11	131	178640	418.3	5	80	104864	332.8
styr	9	5	10	15	413	547056	1352.4	8	217	287216	809.9
s1	8	5	6	11	167	217616	638.3	5	121	156832	475.3
s1a	8	5	6	11	153	199056	568.2	6	96	124816	368.6
s27	4	3	1	4	20	25056	82.7	3	15	18096	60.7
s386	7	7	6	13	123	158688	333.6	4	83	105328	269.6
tav	4	2	4	6	28	34336	106	4	31	39440	124.3
tbk	6	5	3	8	146	190240	433.6	5	160	207872	520.3
tma	7	5	6	11	219	285824	529.1	5	130	169824	392
b01	2	5	2	7	51	58928	346.8	4	39	38720	106.9
b02	1	4	1	5	28	30624	222.1	3	15	18096	45.4
b03	4	30	4	34	159	208800	697.1	4	184	239424	748.0
b07	1	49	8	57	448	578608	1665	6	324	421776	1178
b08	9	21	4	25	160	211584	631.1	5	133	172608	446.8
b09	1	28	1	29	162	207872	664.7	7	208	270512	802.2
b10	11	17	6	23	203	264480	763.3	7	177	230608	740.2

TABLE II
PARITY-BASED CED WITH MINIMUM TREE COUNT VERSUS PARITY-BASED CED WITH ENTROPY-DRIVEN TREE SELECTION

Benchmark	Parity-Based CED with Minimum Tree Count					Parity-Based CED with Entropy-Driven Tree Selection				
Circuit Name	Parity Functions	Normalized Entropy	Number of Gates	Area Cost	Power Consumption	Parity Functions	Normalized Entropy	Number of Gates	Area Cost	Power Consumption
cse	5	0.641	131	171680	454.1	5	0.598	97	127136	372
donfile	4	0.716	57	74704	247.9	4	0.692	56	75632	247.4
dk16	6	0.953	323	428736	1182.2	6	0.883	294	389760	1088.8
dk512	4	0.973	79	104400	340	6	0.818	61	80272	262.6
keyb	5	0.512	82	107648	325.3	5	0.439	67	86304	264.2
pma	6	0.247	186	243136	644.2	6	0.210	138	180496	290.5
sse	5	0.771	80	104864	332.8	5	0.753	59	77488	252.4
styr	8	0.463	217	287216	809.9	8	0.394	162	213904	662.2
s1	5	0.680	121	156832	475.3	5	0.512	61	77024	252.9
s1a	6	0.533	96	124816	368.6	7	0.314	61	77024	226.5
s27	3	0.952	15	18096	60.7	3	0.813	7	8352	42.4
s386	4	0.420	83	105328	269.6	4	0.400	72	92336	226.8
tav	4	0.723	31	39440	124.3	4	0.685	26	33408	109.9
tbk	5	0.683	160	207872	520.3	5	0.647	151	198592	500.3
tma	5	0.258	130	169824	392	7	0.224	131	172144	395.3
b01	4	0.479	39	48720	106.9	5	0.424	25	31088	66.8
b02	3	0.361	15	18096	45.4	3	0.352	9	11600	33.7
b03	4	0.803	184	239424	748.0	4	0.688	133	172608	523.4
b07	6	0.661	324	421776	1178	6	0.474	257	334080	1010.9
b08	5	0.594	133	172608	446.8	5	0.472	98	127600	361.8
b09	7	0.664	208	270512	802.2	7	0.527	118	150336	441.6
b10	7	0.726	177	230608	740.2	7	0.618	136	177712	540.9

number of primary outputs. Under the third major heading, we provide the results of duplication-based CED: number of parity functions; number of gates; hardware cost reported by SIS in λ^2, where λ is the smallest feature size; and power consumption in microwatts. Under the fourth major heading, the same information is reported for parity-based CED with minimum-tree-count selection. The average reduction in parity function count over all benchmark circuits is 52.04%, while the average reduction in area overhead and power consumption is 20.72% and 20.97%, respectively. On several benchmarks, the number of parity functions reduces by more than 50%. For example, this is the case for circuits cse, pma, sse, s1, s386, and tma. However, due to the reasons discussed in Section IV, the corresponding reduction in area overhead and power consumption is not always commensurate with the reduction in the number of parity functions. More importantly, in several cases the reduction in the number of prediction functions results in an increase in area and power-consumption overhead. For example, the number of parity functions for dk512 reduces by 43%, while the area overhead and power consumption increase by more than 8%. Similar observations hold for circuits dk16, tav, and tbk. These results corroborate that while parity-based CED outperforms duplication-based CED, tree-selection methods that account for area and power-consumption overhead are necessary.

C. Minimum Count Versus Entropy-Driven Parity-Tree Selection

The results of parity-based CED with the minimum number of parity-trees-selection method and with the entropy-driven-tree-selection method are summarized in Table II. We provide the name

of the circuit under the first major heading. Under the second major heading, we provide the results of parity-based CED with minimum tree count: number of parity functions, normalized entropy, number of gates, hardware cost, and power consumption. Under the third major heading, the same information is reported for parity-based CED with entropy-driven tree selection. The average reduction in area overhead and power consumption over all benchmark circuits is 24.81% and 23.16%, respectively.

On many benchmarks, the proposed entropy-driven-selection method yields a set of parity trees of minimum cardinality $k = k_{\min}$. Yet the selected solution is picked based on a lower entropy and therefore results in a less expensive hardware implementation and a lower power consumption. For example, this is the case for circuits tav, s27, dk16, s1, pma, and s386. In some cases, such as s27 and pma, the overhead of the solution is reduced by more than 50%.

Additionally, the proposed method selects a set of k parity trees with $k > k_{\min}$, if the entropy of the k-bit parity predictor is lower than the entropy of $k_{\min}$-bit parity predictor. For example, on circuit dk512, the proposed method yields a solution with $k = 6$ parity trees and a normalized entropy of 0.818; while selection of the minimum number of parity trees yields a solution with $k_{\min} = 4$ parity trees and a normalized entropy of 0.973. As may be observed, selecting a solution with a higher number of parity trees, yet with a lower entropy, results in a less expensive parity prediction circuit.

Occasionally, the proposed method will fail to provide hardware or power reduction over the solution selecting the minimum number of parity trees. This is, for example, the case for circuit tma, where the entropy of the proposed solution with $k = 7$ parity trees is slightly lower than the entropy of the solution with $k_{\min} = 5$ parity trees, but the implementation of the latter is slightly less expensive and consumes less power. This is attributed to the heuristics employed for exploring the search space of the problem. Additionally, the entropy is a statistical and not an exact metric. Significant entropy differences provide a good indication of the relative circuit complexity. However, the comparison resolution may degrade as the absolute value of the difference between the entropy of two circuits becomes smaller. Definitely, since the selection of the minimum number of parity trees for $k = k_{\min}$ is the starting point for the entropy-driven metropolis filter, one could always keep this as the best solution seen so far. We chose not to do this and report the actual negative results to emphasize that this is a heuristic search based on a statistical metric.

VII. Conclusion

We introduced a nonintrusive method for performing CED in FSMs that compacts the state/output bits of the circuit through parity trees and compares them to the expected correct values that are computed through an on-chip parity predictor. We formulated the problem of identifying the minimal number of required parity bits as an ILP and we devised an algorithm based on randomized rounding to solve it. As demonstrated experimentally, a small number of parity trees is typically sufficient for lossless compaction, thus reducing the number of on-chip parity prediction functions as compared to duplication. However, the area and power-consumption overhead incurred by the parity predictor is not necessarily proportional to the number of functions. To address this discrepancy, we then presented a method that utilizes the entropy of the parity predictor as a potential function for guiding a search algorithm that selects parity trees that incur low overhead. Our methodology takes an important step towards tackling the question of simultaneously achieving lossless compaction and minimizing the overhead of the parity-predictor function in nontrivial ways. Experimental data on benchmark circuits indicate that entropy-driven tree selection yields significant overhead reduction in both area and power consumption as compared to the basic method that selects the minimum number of parity trees.

References

[1] K. Chakrabarty and J. P. Hayes, "Test response compaction using multiplexed parity trees," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 15, no. 11, pp. 1399–1408, Nov. 1996.

[2] O. Sinanoglu and A. Orailoglu, "Space and time compaction schemes for embedded cores," in *Int. Test Conf.*, Baltimore, MD, 2001, pp. 521–529.

[3] S. Mitra and K. Kim, "X-compact: An efficient response compaction technique for test cost reduction," in *Int. Test Conf.*, Baltimore, MD, 2002, pp. 311–320.

[4] M. Goessel and S. Graf, *Error Detection Circuits*. New York: McGraw-Hill, 1993.

[5] N. A. Touba and E. J. McCluskey, "Logic synthesis of multilevel circuits with concurrent error detection," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 16, no. 7, pp. 783–789, Jul. 1997.

[6] S. Almukhaizim, P. Drineas, and Y. Makris, "On concurrent error detection with bounded latency in FSMs," in *Design Automation and Test Europe Conf.*, Paris, France, 2004, pp. 596–601.

[7] K.-T. Cheng and V. D. Agrawal, "An entropy measure for the complexity of multi-output Boolean functions," in *Design Automation Conf.*, Orlando, FL, 1990, pp. 302–305.

[8] N. Shanbhag, "A mathematical basis for power-reduction in digital VLSI systems," *IEEE Trans. Circuits Syst.*, vol. 44, no. 11, pp. 935–951, Nov. 1997.

[9] E. Macii, M. Pedram, and F. Somenzi, "High-level power modeling, estimation and optimization," in *Design Automation Conf.*, Anaheim, CA, 1997, pp. 504–511.

[10] M. Nemani and F. N. Najm, "Delay estimation VLSI circuits from a high-level view," in *Design Automation Conf.*, San Francisco, CA, 1998, pp. 591–594.

[11] M. Nemani and F. Najm, "Towards a high-level power estimation capability," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 15, no. 6, pp. 588–598, Jun. 1996.

[12] V. V. Danilov, N. V. Kolesov, and B. P. Podkopaev, "An algebraic model for the hardware monitoring of automata," *Autom. Remote Control*, vol. 36, no. 6, pp. 984–991, 1975.

[13] J. F. Meyer and R. J. Sundstrom, "On-line diagnosis of unrestricted faults," *IEEE Trans. Comput.*, vol. C-24, no. 5, pp. 468–475, May 1975.

[14] E. S. Sogomonyan, "The design of discrete devices with diagnostics in the course of operation," *Autom. Remote Control*, vol. 31, no. 11, pp. 1854–1860, 1970.

[15] G. Aksenova and E. Sogomonyan, "Design of self-checking built-in check circuits for automata with memory," *Autom. Remote Control*, vol. 36, no. 7, pp. 1169–1177, Jul. 1975.

[16] C. Zeng, N. Saxena, and E. J. McCluskey, "Finite state machine synthesis with concurrent error detection," in *Int. Test Conf.*, Atlantic City, NJ, 1999, pp. 672–679.

[17] P. Raghavan and C. Thompson, "Randomized rounding: A technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987.

[18] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proc. Annu. ACM Symp. Theory Computing*, Washington, D.C., 1984, pp. 302–311.

[19] L. Khachiyan, "A polynomial-time algorithm for linear programming," *Sov. Math.—Dokl.*, vol. 20, no. 1, pp. 191–194, 1979.

[20] E. M. Sentovich *et al.*, "SIS: A system for sequential circuit synthesis," Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, ERL MEMO. No. UCB/ERL M92/41, 1992.

[21] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 13, 1983.

[22] T. Batu, S. Dasgupta, R. Kumar, and R. Rubinfeld, "The complexity of approximating entropy," in *Proc. Annu. ACM Symp. Theory Computing*, Montreal, QC, Canada, 2002, pp. 678–687.

[23] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of state calculation by fast computing machines," *J. Chem. Phys.*, vol. 21, no. 6, pp. 1087–1092, 1953.

[24] H. K. Lee and D. S. Ha, "HOPE: An efficient parallel fault simulator for synchronous sequential circuits," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 15, no. 9, pp. 1048–1058, Sep. 1996.

[25] M. Berkelaar, *Linear Programming Solver*. Software package available from. [Online]. Available: http://www.cs.sunysb.edu/~algorith/