# Self-Adjusting Constrained Random Stimulus Generation Using Splitting Evenness Evaluation and XOR Constraints*

Shujun Deng, Zhiqiu Kong, Jinian Bian, Yanni Zhao
Department of Computer Science and Technology, Tsinghua University
Beijing, China
{dengsj04, kzq07}@mails.tsinghua.edu.cn, bianjn@tsinghua.edu.cn, zhaoyn05@mails.tsinghua.edu.cn

**Abstract—Constrained random stimulus generation plays significant roles in hardware verification nowadays, and the quality of the generated stimuli is key to the efficiency of the test process. In this work, we present a linear dynamic method to guide random stimulus generation by SAT solvers. A splitting simplified Min-Distance-Sum evaluation method and an XOR sampling strategy are integrated in the self-adjusting random stimulus generation framework. The evenness of the split groups is evaluated to find out some uneven parts. Then, random partial solutions for the uneven parts and random XOR constraints for the other inputs are added into constraints to get better distributed stimuli. Experimental results show that our method can evaluate the evenness as well as more complex formulae for stimulus generation, and also confirm that the self-adjusting method can improve the fault coverage ratio by more than $17\%$ averagely with the same number of stimuli.**

## I. Introduction

Constrained random stimulus generation, which is a combination of formal verification and traditional simulation, is widely applied in practical hardware verification [1] [2] [3]. The objective of constrained random stimulus generation is to find some stimuli that can be used to distinguish between a good circuit response and a faulty circuit response. In another word, the goal is to detect defects and to achieve a given fault coverage. Most of constrained random stimulus generation methods are with weighted Binary Decision Diagram (BDD) sampling and random walks [4] [5]. BDD is built with weighted branches and then the algorithm walks randomly from the root to terminals according to the weights. However, BDD is sensitive with the space size.

One solution of the space explosion problem is to use SAT based methods due to the rapid efficiency improvement of SAT solvers in the last decade [6] [7] [8]. The first method used is that described in [9]. The bias of SAT solving is reduced by random pre-assignment of variables. Because the part assignment can influence the subsequent decision and constraint propagation considerably, the final generated solutions are distributed with some randomization. An alternative is to add XOR constraints [10] [11] [12]. It has been proved both in theory and in practice that, a SAT problem with $N > 1$ solutions can be reduced to only one solution (U-SAT) through randomly adding some XOR constraints with success probability $p \geq \frac{1}{4}$ [13]. Plaza *et al.* [12] emphasized an aspect of this theory. By adding a random XOR constraint into a SAT problem, there is a high probability the solution space can be reduced into half. An XOR constraint $x_1 \oplus x_2 \oplus ... \oplus x_n$ ensures the odd polarity of $x_i$, in another word, there are odd number of variables can be assigned with $true$ value. Consequently, a good distribution of solutions can be achieved by adding random XOR constraints. A point to note here is that the work in [12] is an automated coverage directed test method, which analyzes the feedback from circuit simulation to achieve a higher verification coverage. Since both the above methods are static, the quality of the distribution of solutions relies on the randomization of the pre-assignment and the added XOR constraints.

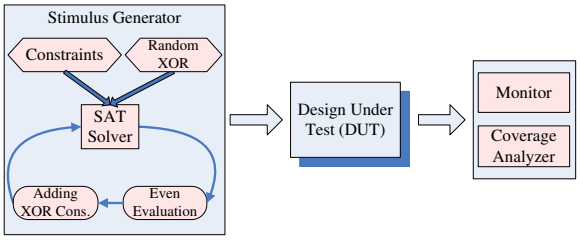



Fig. 1. Self-adjusted random stimulus generation

In this paper, we present a dynamic self-adjusting random stimulus generation framework as shown in Fig.1. SAT solver generates some solutions with user specified constraints and random XOR constraints. Then, these solutions are evaluated to find some most uneven groups. More stimuli will be generated with some XOR constraints for the found uneven groups.

The remainder of this paper is organized as follows: Section II describes the background of even distribution. Even distribution evaluation is analyzed in Section III. Self-adjusting random stimulus generation is presented in Section IV. In Section V, the experiments and comparisons are presented. Section VI concludes the paper.

## II. Background

In this Section, we will give the definitions of Least Even Distribution (LED), Most Even Distribution (MED), and MED

*This work was funded in part by the National Grand Fundamental Research 973 Program of China under Grant No.2005CB321605 and the National Natural Science Foundation of China under Grant No.60876030 and No.90607001.

978-1-4244-2749-9/09/$25.00 ©2009 IEEE

in terms of DFT.

**Definition 1** *Least Even Distribution (LED)*

$K$ solutions selected from $N$-size space are *Least Even Distribution (LED)* when and only when all the selected solutions are the same.

**Definition 2** *Most Even Distribution (MED)*

We define the *Most Even Distribution (MED)* for selecting $K$ solutions from a solution space with $N$ possible solutions based on [14]. The method is as follows:

Assume the $N$ possible solutions are placed evenly on a circle (each with $\frac{2\pi}{N}$ angle) with the index $0, 1, 2, ..., (N-1)$, and the $K$ selected solutions are with the index $S_0, S_1, S_2, ..., S_{K-1}$. $\Delta_i$ is the distance of two solutions which is calculated as:

$$\Delta_i = \begin{cases} S_i - S_{i-1} & \text{if } i \geq 1 \\ S_0 + N - S_{K-1} & \text{if } i = 0 \end{cases}$$

If $\frac{N}{K}$ is an integer, the distribution is an MED if and only if all the $\Delta_i = \frac{N}{K}$. If the division $\frac{N}{K}$ is resulting in a remainder $R$, there are $R$ ($\lfloor \frac{N}{K} \rfloor + 1$) and ($K$–$R$) $\lfloor \frac{N}{K} \rfloor$ in the MED. Now the problem is to distribute $R$ vectors into $K$-size space as evenly as possible. The process is as before with the new $N$ is $K$, and the new $K$ is $R$. In other words, the arguments are $N_0 = N$ and $K_0 = K$ in the first iteration; then, the arguments are $N_1 = K$ and $K_1 = R$, where $R$ is the remainder of $\frac{N}{K}$. This process continues like:

$$\begin{cases} N_i = K_{i-1} \\ K_i = N_{i-1} \% K_{i-1} \end{cases}$$

until $N_x \% K_x = 0$, i.e. the remainder $R$ is 0. Here, '%' stands for the modulo operation.

For example, selecting 6 solutions from a 32-size space is shown in Fig.2. The solution group in Fig.2(a) is MED while that in Fig.2(b) is not.

**Definition 3** *Most Even Distribution (MED) in terms of DFT*

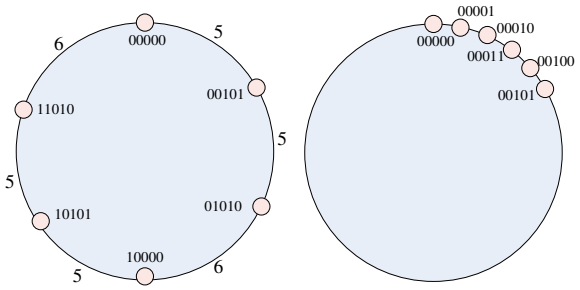
There is another definition in [14] in terms of Discrete Fourier Transform (DFT). In order to be consistent with the random stimulus generation in these work, we give that definition with minor modification.

Assume all the possible solutions (including those do not satisfy the constraints) in the $N$-size space $p_0$=0, $p_1$=1, $p_2$= 2, ..., $p_{N-1} = N-1$ are distributed on a circle, and the selected $K$ solutions are $S_0, S_1, S_2, ..., S_{K-1}$, The sequence $a_1, a_2, a_3, ..., a_N$ in [14] is defined as following:

$$a_i = \begin{cases} 1 & \text{if } p_{i-1} \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

That is the same as:

$$a_i = \begin{cases} 1 & \text{if } \exists\, l.\ (p_{i-1} = S_l, 0 \leq l < K) \\ 0 & \text{otherwise} \end{cases}$$




(a) Most Even Distribtuion (b) Not Most Even Distribtuion

Fig. 2. Most even distribution

The DFT contains frequencies $0, 1, 2, ..., m, ..., (N-1)$:

$$\begin{aligned} F(m) &= \sum_{i=1}^{N} a_i e^{\frac{-j2\pi mi}{N}} \\ &= \sum_{i=0}^{N-1} a_{i+1} e^{\frac{-j2\pi m(i+1)}{N}} \\ &= \sum_{i=0}^{N-1} a_{i+1} e^{\frac{-j2\pi m(p_i+1)}{N}} \\ &= \sum_{i=0}^{K-1} e^{\frac{-j2\pi m(S_i+1)}{N}} \end{aligned} \tag{1}$$

where $j$ is the imaginary part in complex number, $m$ is the frequency in DFT ($0 \leq m \leq N-1$). A point to be noted is that only when $p_i$ equals one of the select solution $S_l,\ 0 \leq l < K$, $a_{i+1}$ is 1, otherwise, $a_{i+1}$ is 0.

Since $e^{-j\theta} = \cos\theta - j\sin\theta$, a distribution is an MED when and only when the formula

$$\begin{aligned} &\sum_{m=1}^{N-1} \frac{|F(m)|^2 \left(m - \frac{N}{2}\right)^2}{N-1} \\ &= \sum_{m=1}^{N-1} \frac{\left| \sum_{i=0}^{K-1} e^{\frac{-j2\pi m(S_i+1)}{N}} \right|^2 \left(m - \frac{N}{2}\right)^2}{N-1} \\ &= \sum_{m=1}^{N-1} \frac{\left| \sum_{i=0}^{K-1} \left( \cos\left(\frac{2\pi m(S_i+1)}{N}\right) - j\sin\left(\frac{2\pi m(S_i+1)}{N}\right) \right) \right|^2 (m - \frac{N}{2})^2}{N-1} \end{aligned} \tag{2}$$

reaches its minimum value.

The formula has been validated by some experiments [14], however, to our best knowledge, there are no formal mathematical proofs yet.

## III. Even Distribution Evaluation

We have known what are LED and MED so far, but how to evaluate the evenness of two different groups of solutions if neither of them is LED or MED in practice? To achieve this goal, we will present some practically good methods to evaluate evenness in constrained random stimulus generation.

In our opinion, it is quite difficult to give a general definition of evenness. For example, given a problem with $N = 27$,

$K = 6$, we know that $\Delta$={4, 5, 4, 5, 4, 5} is MED. However, is {4, 4, 6, 4, 4, 5} better or worse than {4, 4, 4, 5, 5, 5}? So the definition of evenness has to be combined with the application environment. In this paper, we focus on random stimulus generation. In the following subsections, we will give some effective evenness evaluation formulae that can be used to measure the evenness degree of the solutions for random stimulus generation.

### A. Weighted Min-Distance-Sum Evaluation

Assume the size of the solution space is still $N$ and the selected $K$ solutions are $S_0, S_1, S_2, ..., S_{K-1}$ whose values are the index in the solution space (in the range $[0, N-1]$). A better evaluation formula named min-distance-sum for evenness degree is as follows:

$$\sum_{u=1}^{K-1}\left(\left(\sum_{i=0}^{K-1}\left|\frac{N}{K}-\frac{\Delta_{i_u}}{u}\right|\right)^2(u-\tfrac{N}{2})^2\right) \tag{3}$$

Here,

$$\Delta_{i_u}=\begin{cases} S_i - S_{i-u} & \text{if } i \geq u \\ S_i + N - S_{i+K-u} & \text{otherwise}\end{cases}$$

In the formula (3), we calculate the distance between each solution and the previous one, where "previous" here is different according to the actual value of $u$. Actually, "previous" means $u$ steps before it on a clockwise circle. What's more, different weights $(u-\frac{N}{2})^2$ are multiplied to the distances sum according to $u$ considering that doing so will empirically improve the rationality of the evaluation value.

In order to compare the results easily, we normalize the formula (3) with a constant, which is the value of the formula (3) when the distribution is LED. The new formula is as following:

$$\frac{\sum_{u=1}^{K-1}\left(\left(\sum_{i=0}^{K-1}\left|\frac{N}{K}-\frac{\Delta_{i_u}}{u}\right|\right)^2(u-\frac{N}{2})^2\right)}{\frac{N^2}{K^2}\sum_{u=1}^{K-1}((K-u)^2(2u-N)^2)} \tag{4}$$

When the distribution is LED, the value of the formula (4) is 1. LED implies that all the solutions are the same, then $\Delta_{0_u} = \Delta_{1_u} = \cdots = \Delta_{(u-1)_u} = N$, $\Delta_{(u)_u} = \Delta_{(u+1)_u} = \cdots = \Delta_{(k-1)_u} = 0$ according to the definition of $\Delta_{(i)_u}$.

### B. Simplified Min-Distance-Sum Evaluation

The formulae (2) and (4) is precise for evenness evaluation; however, the calculations for them are time-consuming. Their complexities are $O(NK)$ and $O(K^2)$ independently. In random stimulus generation, there is no need to find out the most evenly distributed solutions. Therefore, we present a new, simplified evaluation method as follows:

$$\frac{\sum_{i=0}^{K-1}\left|\frac{N}{K}-\Delta_{i_1}\right|}{2\frac{K-1}{K}N} \tag{5}$$

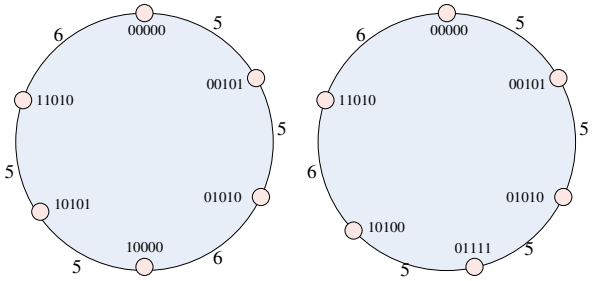



Fig. 3. 1-step most even distribution

$$\Delta_{i_1}=\begin{cases} S_i - S_{i-1} & \text{if } i \geq 1 \\ S_0 + N - S_{K-1} & \text{if } i = 0\end{cases}$$

The complexity of the above formula is $O(K)$. Actually, the formula (5) only tests 1-step distance while the formula (4) tests from 1-step distance to ($K$-1)-step distance with different weights. Consequently, the formula (5) cannot distinguish the distributions in Fig.3. Actually, only the left part in Fig.3 is MED.

**Theorem 1** *The maximum value of the formula (5) is 1, and it gets the maximum value when all the solutions are the same (LED).*

The detailed proof is omitted for the page limit.

## IV. Self-Adjusting Random Stimulus Generation

Given a Design Under Test (DUT), random stimulus generation is to select $K$ solutions from $N$-size space to find out as many defects as possible. The goal is to achieve a high coverage. Static methods generate all stimuli one by one with directive constraints or random assignments. The self-adjusting random stimulus generation presented here is a dynamic framework, in which the current generated stimuli can guide subsequent generations. This can be completed by splitting even distribution evaluation and XOR constraints.

The pseudo-code of self-adjusting random stimulus generation is shown in Fig.4. We generate $K$ stimuli in following steps: $\left\lfloor\frac{K}{t}\right\rfloor, \left\lfloor\frac{t-1}{t}\left\lfloor\frac{K}{t}\right\rfloor\right\rfloor, \left\lfloor\frac{t-1}{t}\left\lfloor\frac{t-1}{t}\left\lfloor\frac{K}{t}\right\rfloor\right\rfloor\right\rfloor, \ldots\ldots$ until the number is less than $\left\lfloor\frac{K}{s}\right\rfloor$. Consequently, we can analyze the complexity of the algorithm in Fig.4 simply. Assume the loop time of $while$ is $T$, then $\left\lfloor\frac{t-1}{t}\left\lfloor\frac{t-1}{t}\left\lfloor\frac{t-1}{t}\left\lfloor\frac{t-1}{t}\left\lfloor\frac{t-1}{t}\left\lfloor\frac{K}{t}\right\rfloor\right\rfloor\right\rfloor\right\rfloor\right\rfloor\right\rfloor \leq \left(\frac{t-1}{t}\right)^T\left(\frac{K}{t}\right) \leq \frac{K}{s}$, so $T \leq log_{(\frac{t-1}{t})}\frac{t}{s}$. The parameters $t$ and $s$ can be configured. We use $t$=3 and $s$=16 for the experiments in Section V. Since the complexity of the Split-Evaluation function $split\_evaluate$ is O($K$), the main cost of the algorithm is still the stimulus generation by SAT solver, especially for difficult cases.

As shown in Fig.5, we split the current stimuli (with $n$-bit width) into groups, each with $\lceil\log_2 cur\_sti\rceil$ width. The reason why we select $\lceil\log_2 cur\_sti\rceil$ as the width is to unify the even distribution evaluation and the effect of stimuli in random stimulus generation. If the width is more than $\lceil\log_2 cur\_sti\rceil$,

**Inputs:**
[1] bit width $n$;
[2] number of expected stimuli $K$ ($K > 16$);
[3] inputs $Ins$;
[4] circuit and specified constraints $Cons$.

**Output:** $K$ generated stimuli

**Algorithm:**

```
1. cur_sti ⇐ 0;                    /* current stimuli */
2. inc_sti ⇐ ⌊K/t⌋;    /* stimuli generated each time */
3. while ( cur_sti < K ) {
4.    if (inc_sti ≤ ⌊K/s⌋) { /* the last time */
5.       /* generate all the left stimuli */
6.       inc_sti = K − cur_sti;
7.    }
8.    if ( inc_sti ≠ ⌊K/t⌋ ) { /* not the first time */
9.       split_evaluate();
10.   }
11.   else {
12.      /* generate inc_sti stimuli */
13.      for ( i = 0; i < inc_sti; i++ ) {
14.         Add random XOR constraints for Ins;
15.         Generate one stimulus using Minisat;
16.      }
17.   }
18.   cur_sti += inc_sti;
19.   /* the number of stimuli generated next time*/
20.   inc_sti = ⌊inc_sti * (t−1)/t⌋;
21. }
22. return stimuli;
```

Fig. 4. Algorithm for self-adjusting random stimulus generation

it will incorrectly assume this {000000, 010000, 100000, 110000} as MED. In fact, the Split-Evaluation value (the sum of evaluation values for all groups) in this case is quite bad. The reason is that each group is with $\lceil \log_2 4 \rceil$=2 bit width, and the last two groups are LED. Also, if $\log_2 cur\_sti$ is an integer, the MED of $cur\_sti$ solutions with $\log_2 cur\_sti$ width is from 0 to $cur\_sti$-1, i.e. each solution occurs one time. When $\log_2 cur\_sti$ is not an integer, the case is more or less the same. If the width is less than $\lceil \log_2 cur\_sti \rceil$, there will be many redundant cases for the splitting groups. For example, assume $cur\_sti$=16, and we select the width as 2, then for each group, there are only four different cases: 00, 01, 10, 11 for 16 solutions.

After an initial generation ($\lfloor \frac{K}{t} \rfloor$ in the algorithm), we use the formula (5) to evaluate every group, and select the most inactivated groups. Then, directive partial solution and XOR constraints are generated. For example, when the most inactivated groups have a width of 4 bits, a random value of 0 to 15 will be generated as the partial solution. $\lceil \log_2 inc\_sti \rceil$ random XOR constraints are generated for the other inputs. $inc\_sti$ is the number of the new stimuli to be generated for each iteration, starting from $\lfloor \frac{K}{t} \rfloor$, and is multiplied by $\frac{t-1}{t}$ each time. This process repeats until the last time when $inc\_sti \leq \lfloor \frac{K}{s} \rfloor$. At that time, we assign the number of the left stimuli ($K$–$cur\_sti$) to $inc\_sti$.

**Function:**

```
1. Function split_evaluate() {
2.    /* split the current stimuli into groups, each with
         ⌈log2 cur_sti⌉ size. */
3.    split();
4.    /* evaluate each group using the formula (5)
         independently, recorded into the array a_evalu. */
5.    a_evalu = simp_mds();
6.    sort(a_evalu);
7.    select_the_worst_group();
8.    /* generate inc_sti stimuli */
9.    for ( i = 0; i < inc_sti; i++ ) {
10.      Add random solution for the worst group;
11.      Add random XOR constraints for other Ins;
12.      Generate one stimulus using Minisat;
13.   }
14. }
```

Fig. 5. Split-Evaluation function

If the problem has no solution with the added partial solution and XOR constraints, we has a metric to change for the other groups. If it fails after several tries, we will restart the process.

## V. Experimental Results

### A. Comparison of Different Evaluation Methods

The first experiment is to compare the evenness evaluation formulae (2), (4) and (5). When computing using the formula (2), we normalize it by dividing the value of LED. We randomly generate some solutions with some disturbing strategies to reduce the locality of pseudo-random algorithm. Therefore, the generated solutions are quite different. After the random generation process, we evaluate the solutions using these three formulae independently. Finally sort the arrays with ascending order according to the formula (4). This experiment completed on an Intel Xeon 3GHz CPU with 6G RAM. The results are shown in Fig.6, in which 'MDS' stands for the formula (4), and 'Simp-MDS' is the formula (5) while 'DFT' represents the formula (2). The solid lines stands for the first case with the size of the solution space $N$=256 while the dot dash lines stands for the second one with $N$=65536. $K$ is 32 for both cases. The average evaluation time in seconds is marked on each line. We can see that, the average evaluation time for formula (5) is 0.003s for both cases. The average evaluation time for formula (4) is 0.300s when $N$ = 256 and is 0.290s when $N$=256. Those for formula (2) are 3.585s and 4.598s respectively. This confirms that the evaluation time of the formula (5) is linear to $K$, irrelative to the size of the solution space $N$. The trends for the three formulae are similar. They are due to the different definitions of evenness used in these formulae. For constrained random stimulus generation, the simplified Min-Distance-Sum formula is quite good enough.

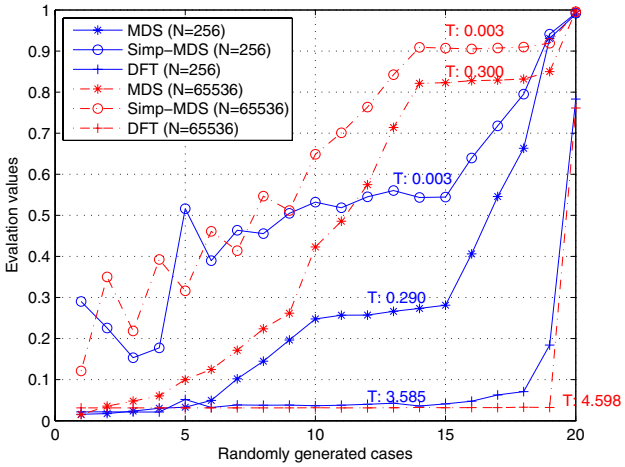



Fig. 6. Comparison of different evaluation methods ($K$ = 32)

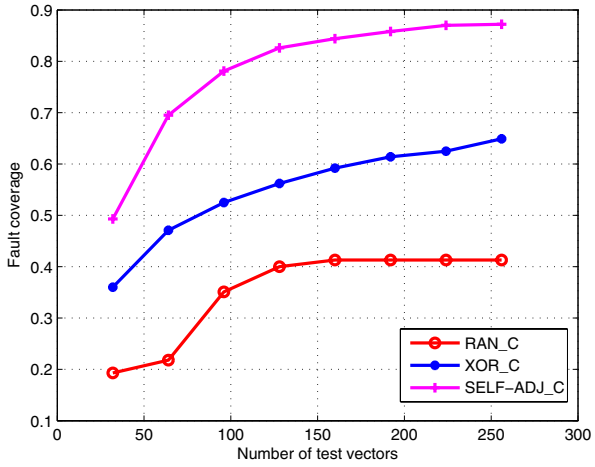



Fig. 7. Results for s27 expanded for 50 time-frames with $K$ from 32 to 256

### *B. Guiding Random Stimulus Generation*

The other experiment uses the fault simulator HOPE [15] to evaluate the quality of the generated stimuli. The benchmarks are the ISCAS89 Sequential Circuits [16], which have been expanded for some time-frames like bounded model checking (BMC) [17] and then translated into Conjunctive Normal Form (CNF). All these transformations are completed by Perl script in linear time. The specified constraints in our experiments are added randomly. The CNF are solved by Minisat 2 [8]. According to the characteristic of combinational circuits, the sizes of the solution spaces for these combinational circuits are $N \leq 2^{n_{in}}$, where $n_{in}$ is the number of primary inputs. All the experiments completed on an Intel Xeon 2.8GHz CPU with 1G RAM. Considering the randomization, we run each benchmark 10 times independently and get the average CPU time and fault coverage ratio values at last.

To estimate the relationship between the coverage ratio and the number of test vectors for different methods, we used the benchmark s27.bench in ISCAS89 which was expanded for 50 time-frames. The results are shown in Fig.7. The X-axis is the number of expected stimuli $K$ (from 32 to 256) and Y-axis stands for the coverage ratios. 'XOR' method is like that in [12] which adds XOR constraints randomly to get an even distribution; 'RAN' method is a direct random method by adding blocking clauses to prevent reduplicate solutions, 'SELF-ADJ' is our self-adjusting stimulus generation method. The postfix '_C' stands for the coverage ratio. We found that the coverage ratio of 'SELF-ADJ' was the highest for the same number of test vectors. Of course, for all the methods, more faults are found when the number of test vectors increase.

The experimental results for some circuits in ISCAS89 are listed in Table. I. The columns are: the name of the test-case in ISCAS89 following with the number of expanded time-frames; the number of the collapsed faults [15] considered in that test-case; the number of expected stimuli $K$; the fault coverage ratio of 'RAN'; the run time used by 'RAN'; the same for 'XOR' and 'SELF-ADJ'. The run time contains the solving time by Minisat and the evaluation time (only for 'SELF-ADJ'). The average data are shown in the last row. With the same number of stimuli, the average fault coverage ratio of 'SELF-ADJ' is quite more than other methods, while the run time is a little more than that of 'XOR' because it has to calculate the splitting evenness and select the worst groups. Because those test-cases like s298_5 are too simple for Minisat, the solving time is considerably less than the evaluation time; however, for those test-cases like s15850_2, the evaluation time can be ignored when comparing with the solving time. As discussed in Section III, the evaluation is linear to the number of the expected stimuli $K$, while the constraint solving is with exponential complexity. Therefore, 'SELF-ADJ' can improve the fault coverage ratio considerably for hard test-cases while the run time is more or less the same as random stimulus generation.

## VI. Summary and Conclusions

We have analyzed different evenness evaluation strategies and proved the simplified Min-Distance-Sum is adequate for applications in constrained random stimulus generation, and also presented a self-adjusting framework with Split-Evaluation and random XOR constraints. By adjusting the splitting evenness dynamically, the stimuli generated are better distributed. Experimental results confirm that the simplified Min-Distance-Sum is good enough for evenness evaluation, and most importantly, the evaluation time is only linear to the number of the expected stimuli. The second experiment shown that the self-adjusting method can improve the fault coverage ratio considerably with the same number of stimuli. This confirms that the proposed method can get better distribution.

In the future, we will make further efforts to optimize the algorithm. What's more, in order to estimate the power of this self-adjusting framework in practical random stimulus generation, we need to do more experiments with other fault models and practical circuits.

TABLE I
EXPERIMENTAL RESULTS FOR ISCAS89 CIRCUITS

| Test-case | #Faults | $K$ | RAN | | XOR | | ROW | |
|---|---|---|---|---|---|---|---|---|
| | | | #RAN_C | RAN_T | #XOR_C | XOR_T | #SELF-ADJ_C | SELF-ADJ_T |
| s298_5 | 1428 | 32 | 33.40% | 45.77 | 56.03% | 16.40 | 67.42% | 73.03 |
| | | 64 | 37.74% | 155.45 | 67.51% | 44.01 | 81.94% | 129.55 |
| s382_5 | 1827 | 32 | 28.52% | 55.66 | 33.59% | 12.51 | 73.89% | 56.67 |
| | | 64 | 29.23% | 121.13 | 43.59% | 32.47 | 81.74% | 134.35 |
| s386_5 | 1872 | 32 | 35.68% | 41.99 | 43.00% | 17.48 | 52.60% | 41.61 |
| | | 64 | 40.55% | 93.31 | 49.95% | 31.87 | 62.46% | 133.17 |
| s1196_2 | 2439 | 32 | 46.49% | 9.23 | 45.80% | 8.99 | 52.98% | 13.21 |
| | | 64 | 54.70% | 19.77 | 55.66% | 17.91 | 64.27% | 28.72 |
| s1238_2 | 2665 | 32 | 40.79% | 8.89 | 45.04% | 9.10 | 49.34% | 10.18 |
| | | 64 | 48.70% | 17.36 | 54.93% | 17.21 | 61.34% | 29.28 |
| s1488_2 | 2960 | 32 | 48.28% | 229.30 | 39.06% | 240.17 | 53.67% | 325.54 |
| | | 64 | 53.21% | 595.87 | 45.01% | 555.77 | 69.36% | 1066.30 |
| s1494_2 | 3000 | 32 | 50.93% | 280.73 | 39.55% | 232.02 | 54.17% | 218.42 |
| | | 64 | 55.83% | 629.55 | 42.13% | 554.16 | 67.81% | 779.41 |
| s13207_2 | 18292 | 32 | 52.93% | 1704.01 | 53.45% | 1527.48 | 53.99% | 1433.44 |
| | | 64 | 56.85% | 2937.05 | 59.42% | 3428.06 | 60.60% | 3303.68 |
| s15850_2 | 22256 | 32 | 48.63% | 2271.92 | 49.52% | 2094.03 | 54.01% | 2077.91 |
| | | 64 | 52.51% | 5559.34 | 56.39% | 4912.75 | 61.22% | 4849.82 |
| Average | 6304.3 | 48 | 45.28% | 820.91 | 48.87% | 764.02 | 62.38% | 816.91 |


## ACKNOWLEDGEMENTS

The authors would like to thank Prof. Kwang-Ting (Tim) Cheng, and the members in his group, and the reviewers for helpful suggestions.


## REFERENCES


[1] Synopsys Inc., "Constrained-random test generation and functional coverage with Vera," *Technical report*, Feb. 2003.

[2] L. Singh, L. Drucker, "Advanced Verification Techniques: A SystemC Based Approach for Successful Tapeout," *Springer US*, 2004.

[3] SystemC Verification Working Group, "Systemc verification standard specification," *OSCI website: http://www.systemc.org*, May. 2003.

[4] J. Yuan, A. Aziz, C. Pixley, K. Albin, "Simplifying Boolean constraint solving for random simulation-vector generation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, No. 3, pp. 412–420, 2004.

[5] J. Yuan, C. Pixley, A. Aziz, "Constraint-Based Verification," *Springer US*, Jan. 2006.

[6] J. P. Marques-Silva and K. A. Sakallah, "GRASP-A New Search Algorithm for Satisfiability," *IEEE/ACM international conference on Computer-aided design (ICCAD)*, pp. 220–227, 1997.

[7] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," *the 38th Design Automation Conference (DAC)*, pp. 530–535, 2001.

[8] N. Eén and N. Sörensson, "An extensible SAT solver," *International Conference on Theory and Applications of Satisfiability Testing*, pp. 502–518, 2003.

[9] N. Kitchen, A. Kuehlmann, "Stimulus Generation for Constrained Random Simulation," *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 258–265, 2007.

[10] C. P. Gomes, A. Sabharwal, B. Selman, "Near-Uniform Sampling of Combinatorial Spaces Using XOR Constraints," *the 20th Annual Conference on Neural Information Processing Systems*, pp. 481–488, Dec 2006.

[11] C. P. Gomes, W. Hoeve, A. Sabharwal, B. Selman, "Counting CSP Solutions Using Generalized XOR Constraints," *the 22nd Conference on Artificial Intelligence*, pp.204–209, July 2007.

[12] S. M. Plaza, I. L. Markov, and V. Bertacco, "Random Stimulus Generation using Entropy and XOR Constraints," *the conference on Design, Automation and Test in Europe*, pp. 664–669, 2008.

[13] L. G. Valiant and V. V. Vazirani, "NP is as easy as detecting unique solutions," *the 17th annual ACM symposium on Theory of computing*, pp. 458–463, 1985.

[14] A. J. Compton, "An Algorithm for the Even Distribution of Entities in One Dimension," *the Computer Journal*, Vol. 28, No. 5, pp. 530–537, 1985.

[15] H. K. Lee and D. S. Ha, "HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 15, No. 9, pp. 1048- 1058, 1996.

[16] ISCAS89 Sequential Benchmark Circuits: http://www.ece.vt.edu/mhsiao/iscas89.html

[17] E. Clarke, A. Biere, R. Raimi and Y. Zhu, "Bounded Model Checking Using Satisfiability Solving," *Formal Methods in System Design*, vol. 19, No. 1, pp. 7–34, 2001.