14th Asia and South Pacific Design Automation Conference

ASP-DAC 2009

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

bianjn@tsinghua.edu.cn

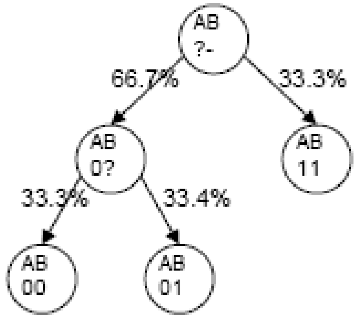
Introduction

- Functional Verification: Bottleneck
- Formal Verification
  - ✓ Completeness ✘ Scalability
- Simulation
  - ✘ Completeness ✓ Scalability
  - Constrained random stimulus generation
    - An enhancement to traditional simulation
    - To find stimuli that would result in different responses for the good and the erroneous circuits

Introduction

- Constrained random stimulus generation
  - Weighted BDD sampling + random walks
    - SystemC Verification Library

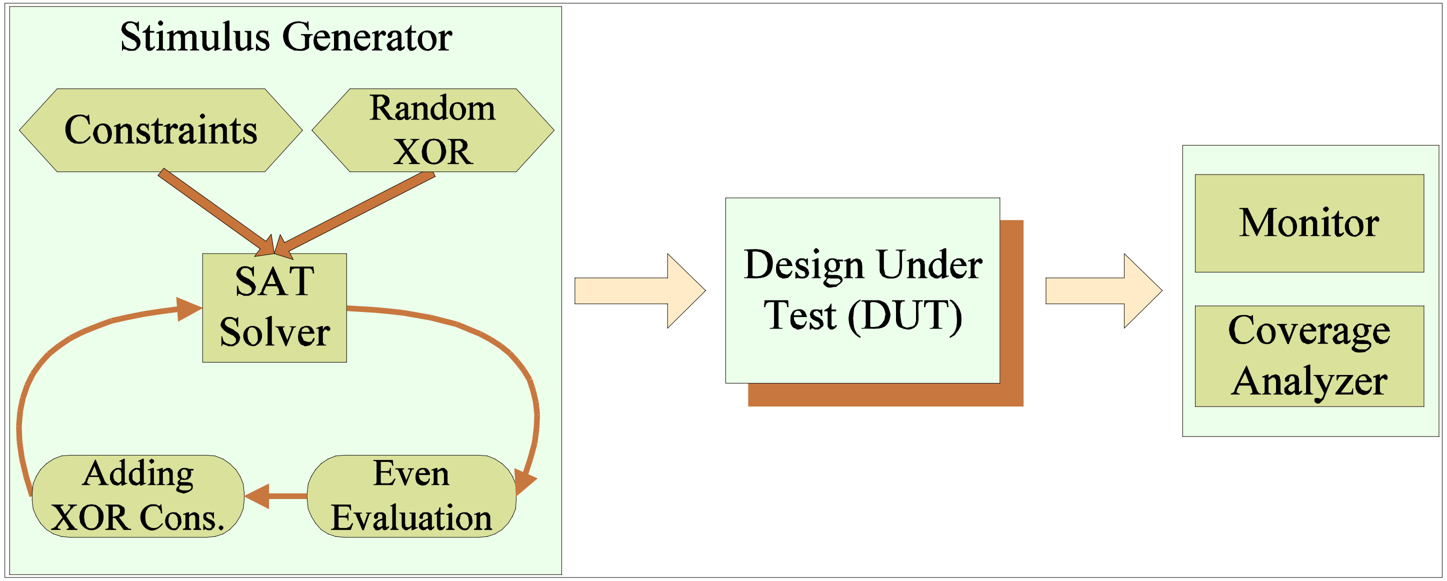
Solution	Number of Hits	Probability
{0,0}	3403	34.03%
{0,1}	3267	32.67%
{1,0}	0	0%
{1,1}	3330	33.3%



- ✘ ✘ Scalability
  - SAT-based method
    - Pre-assignment
      - Assign values to randomly selected variables
    - XOR constraints
      - Adding XOR constraints for randomly selected variables

Introduction

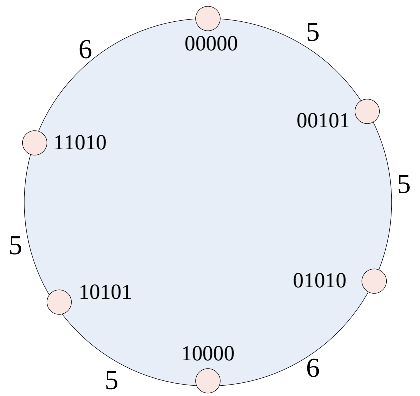
- Our solution



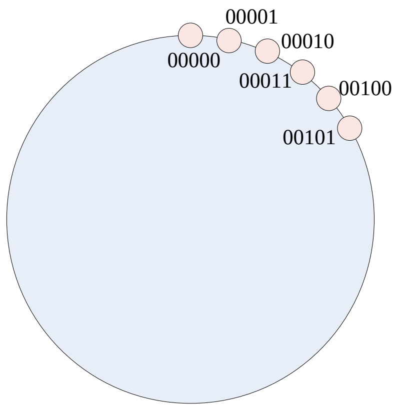
  - Dynamic: self-adjusting
  - Irrelative with the detailed design: no coverage feedback

Background

- Definitions
  - Problem: Distribution of *K* solutions selected from *N*-sized space
  - Least Even Distribution (LED)
    - Solutions are the same
  - Most Even Distribution (MED)
    - Solutions distributed evenly



(a) Most Even Distribtuion



(b) Not Most Even Distribtuion

Background

- XOR Constraints
  - 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 $p >= 1/4$
    - L. G. Valiant et al., “NP is as easy as detecting unique solutions,” *ACM symposium on Theory of computing*, pp. 458–463, 1985.
  - By adding a random XOR constraint into a SAT problem, there is a high probability the solution space can be reduced into half.
    - S. M. Plaza et al., ”Random Stimulus Generation using Entropy and XOR Constraints,” *DATE*, pp. 664–669, 2008.

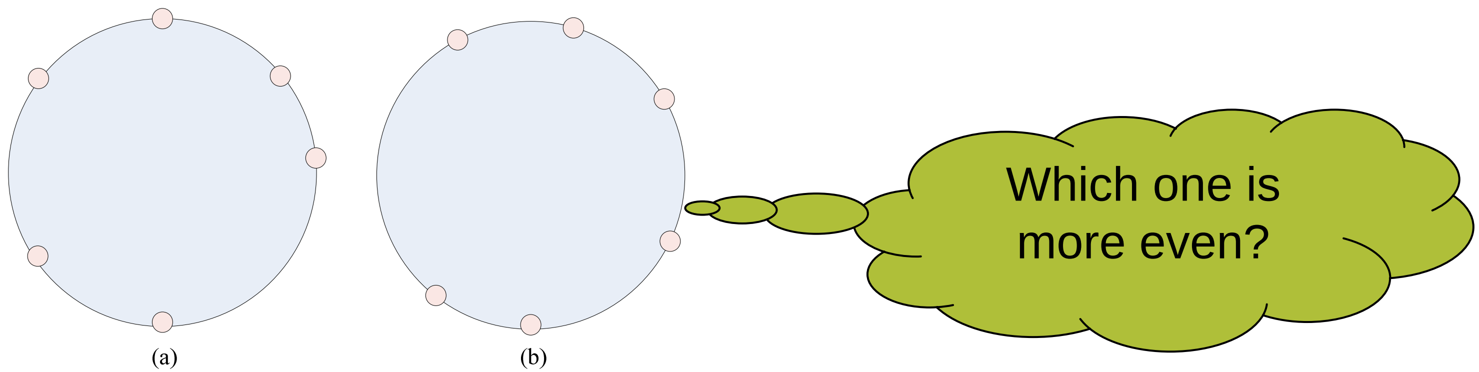
Background

- XOR Constraints
  - Original CNF
    - $(a + b)(b + \sim c + d)(\sim a + c + \sim d)(c + d)$
    - Initial solution space {0101, 0110, 0111, 1011, 1110, 1111}
  - The results for added XOR constraints

XOR Cons.	Final Solutions	#Final Sol. : #Rem. Sol.
$a \oplus b$	{0101, 0110, 0111, 1011}	4 : 2
$a \oplus c$	{0110, 0111}	2 : 4
$a \oplus d$	{0101, 0111, 1110}	3 : 3
$b \oplus c$	{0101, 1011}	2 : 4
$b \oplus d$	{0110, 1011, 1110}	3 : 3
$c \oplus d$	{0101, 0110, 1110}	3 : 3
$a \oplus b \oplus c$	{0101, 1110, 1111}	3 : 3
$a \oplus b \oplus d$	{0110, 1111}	2 : 4
$b \oplus c \oplus d$	{0111, 1111}	2 : 4
$a \oplus b \oplus c \oplus d$	{0111, 1011, 1110}	3 : 3

Even Distribution Evaluation

How to Evaluate the Evenness?



- Evaluating methods
  - Method based on Discrete Fourier Transfer (DFT)
  - Our methods
    - Weighted Min-Distance-Sum
    - Simplified Min-Distance-Sum

Background

- MED in terms of Discrete Fourier Transfer (DFT)

$$\sum_{m=1}^{N-1} \frac{|F(m)|^2 \left(m - \frac{N}{2}\right)^2}{N-1}$$

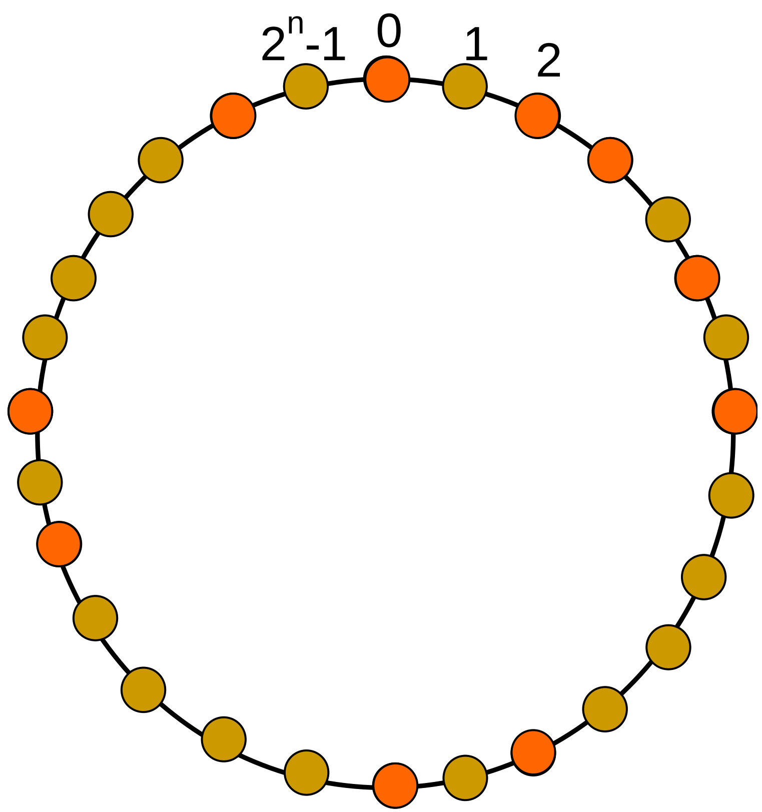
$$= \sum_{m=1}^{N-1} \frac{\left| \sum_{i=0}^{K-1} e^{\frac{-j2\pi m(S_i+1)}{N}} \right|^2 \left(m - \frac{N}{2}\right)^2}{N-1}$$

$$= \sum_{m=1}^{N-1} \frac{\left| \sum_{i=0}^{K-1} \left(\cos\left(\frac{2\pi m(S_i+1)}{N}\right) - j\sin\left(\frac{2\pi m(S_i+1)}{N}\right)\right) \right|^2 \left(m - \frac{N}{2}\right)^2}{N-1}$$

- 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.

Even Distribution Evaluation

- All the possible solutions are located on a circle
- Problem transfer:
  - Evenness →
    - Distance of each adjoining solutions
    - Weighted minimum distance sum



Our Evaluation Methods

- Weighted Min-Distance-Sum

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

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

Our Evaluation Methods (Cont')

- Simplified Min-Distance-Sum

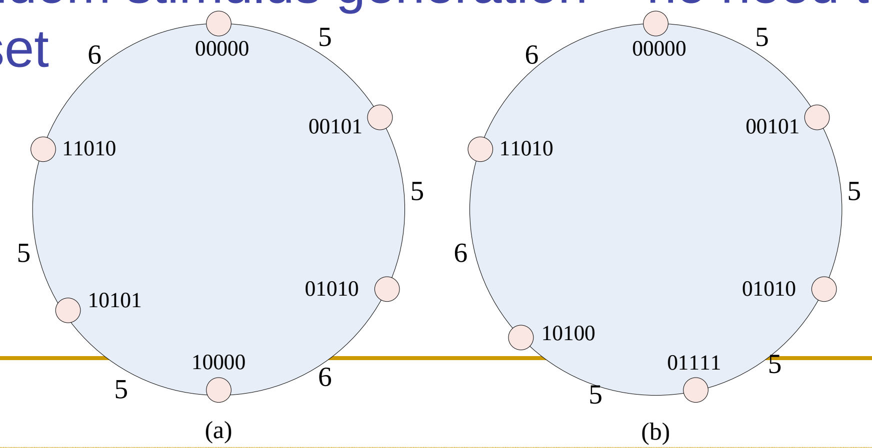
$$D = \frac{\sum_{i=0}^{k-1} \left| \frac{2^n}{k} - \Delta_\mathrm{i} \right|}{\frac{k-1}{k} 2^{n+1}}$$

$$\Delta_\mathrm{i} = \begin{cases} S_i - S_{i-1}, & i = 1, 2, \ldots, k-1 \\ S_0 + 2^n - S_{k-1}, & i = 0 \end{cases}$$

  - *D* ranges from 0 to 1
  - Less *D* means more even distributed

Difference

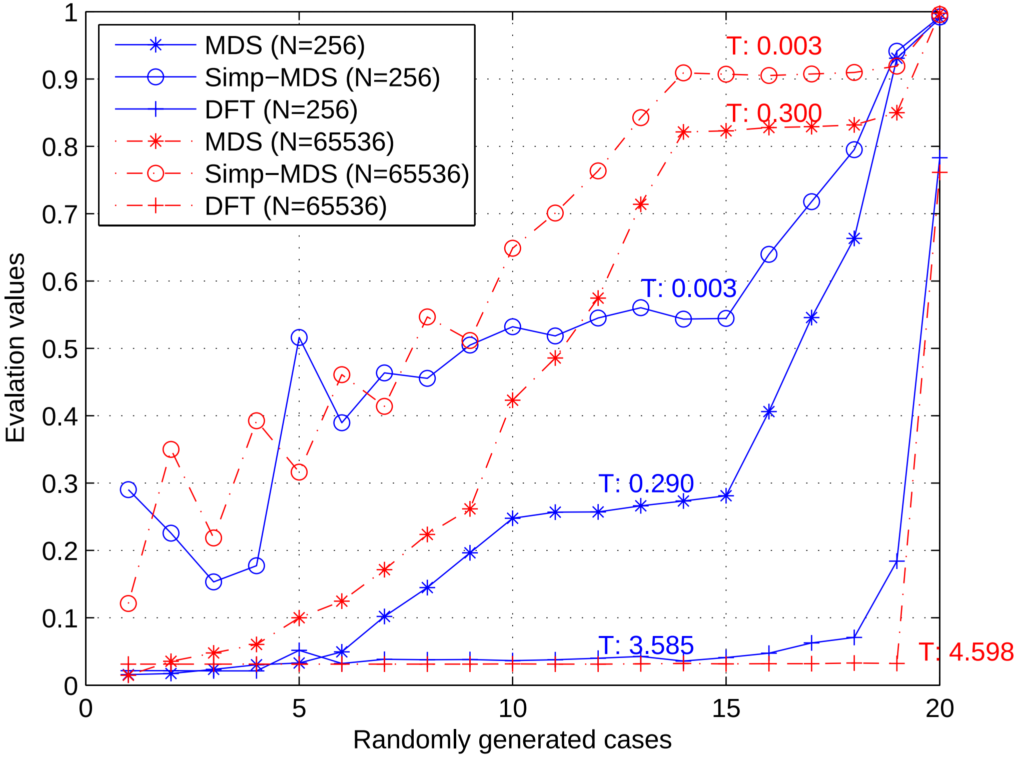
- Weighted Min-Distance-Sum
  - 1-step to (K-1)-step distances
  - *Complexity:* $O(K^2)$
  - Can distinguish {5,5,6,5,5,6} from {5,5,5,5,6,6}
- Simplified Min-Distance-Sum
  - Only 1-step distance
  - *Complexity:* $O(K)$
  - Can not distinguish {5,5,6,5,5,6} from {5,5,5,5,6,6}
  - That is all right for random stimulus generation – no need to find the optimal stimulus set



Experiments for Evaluation

- Simplified Min-Distance-Sum (Simp-MDS)
  - Time: (1) 0.003s (2) 0.003s
- Weighted Min-Distance-Sum (MDS)
  - Time: (1) 0.290s (2) 0.300s
- MED in terms of Discrete Fourier Transfer (DFT)
  - Time: (1) 3.585s (2) 4.598s

- The trends are similar for different test-cases



- Conclusion: Simp-MDS is more efficient than other methods, and it is adequate for random stimulus generation

Self-Adjust Framework based on Simplified Min-Distance-Sum

Limitation of Evenness Evaluation

- This stimulus's evenness score is good, but it is not good for practical simulation
- Splitting strategy is needed

$$K\begin{cases}00000000000\\01000000000\\10000000000\\11000000000\end{cases}$$

Splitting Strategy

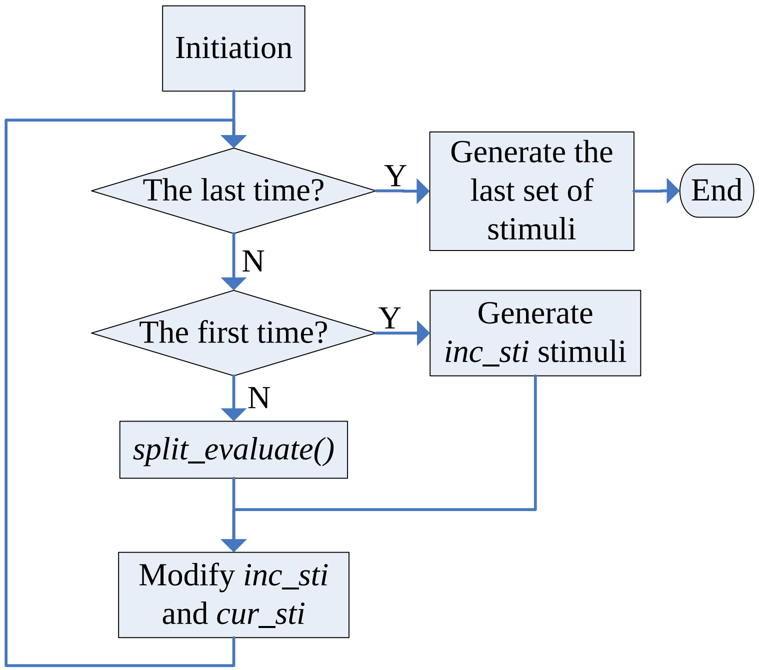
- K is the expected number of test vectors
- Split each test vector into groups with $\log_2 K$ width

$$K\begin{cases} 01\ldots10\;01\ldots11\ldots\ldots00\ldots01\ldots \\ 11\ldots11\;00\ldots10\ldots\ldots10\ldots11\ldots \\ \ldots\ldots \\ \underbrace{\underbrace{00\ldots01}_{\log_2 K}\;\underbrace{11\ldots10}_{\log_2 K}\ldots\ldots\underbrace{01\ldots10}_{\log_2 K}\ldots}_{n} \end{cases}$$

Main Framework

Algorithm:

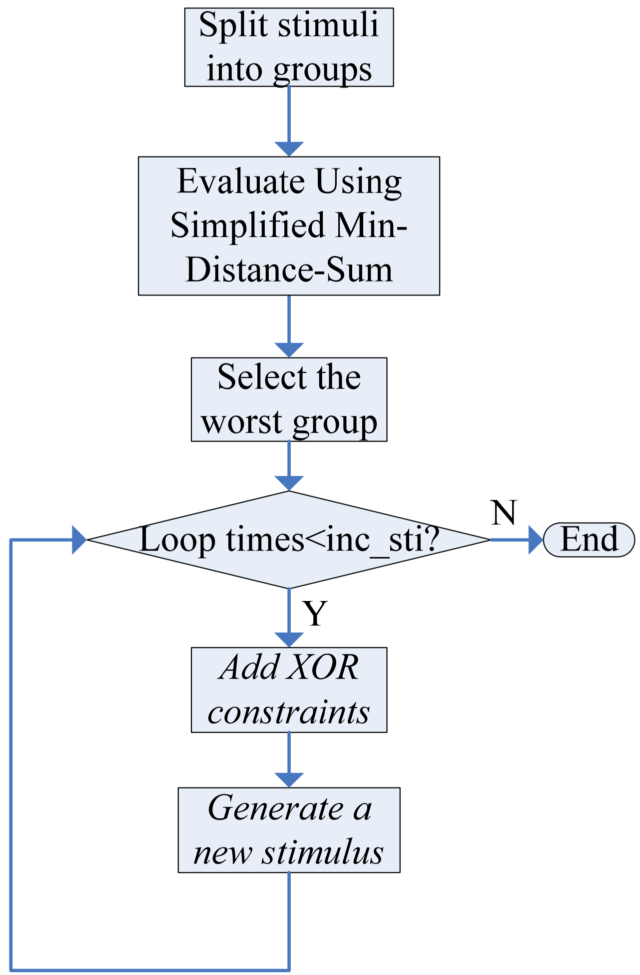
$cur\_sti \Leftarrow 0;$ /* current stimuli */
$inc\_sti \Leftarrow \lfloor \frac{K}{t} \rfloor;$ /* stimuli generated each time */
while ($cur\_sti < K$) {
  if ($inc\_sti \leq \lfloor \frac{K}{s} \rfloor$) { /* the last time */
    /* generate all the left stimuli */
    $inc\_sti = K - cur\_sti;$
  }
  if ($inc\_sti \neq \lfloor \frac{K}{t} \rfloor$) { /* not the first time */
    $split\_evaluate();$
  }
  else {
    /* generate inc_sti stimuli */
    for ($i = 0;\ i < inc\_sti; i++$) {
      Add random XOR constraints for Ins;
      Generate one stimulus using Minisat;
    }
  }
  $cur\_sti \mathrel{+}= inc\_sti;$
  /* the number of stimuli generated next time*/
  $inc\_sti = \lfloor inc\_sti * \frac{t-1}{t} \rfloor;$
}
return $stimuli;$



Split-Evaluation Function

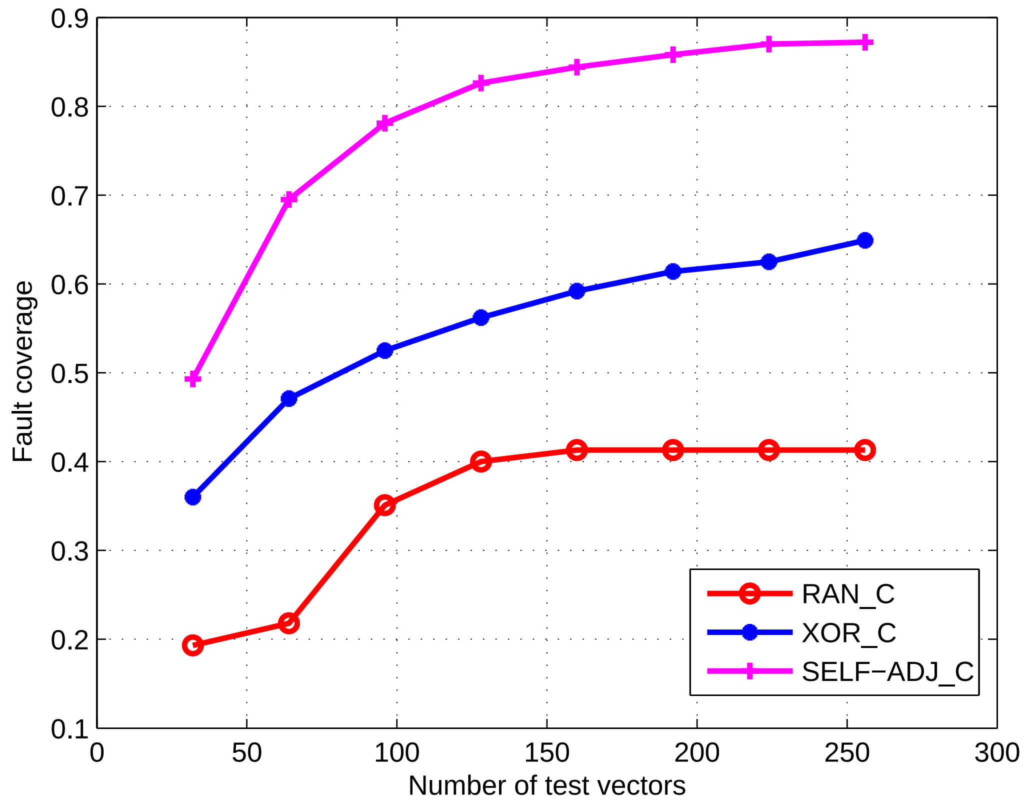
Function:

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



Experimental Results

- Benchmark: s27 in ISCAS89 expanded for 50 time-frames
  - RAN: direct random stimulus generation
  - XOR: stimulus generation with random XOR constraints
  - SELF-ADJ: the self-adjusting method

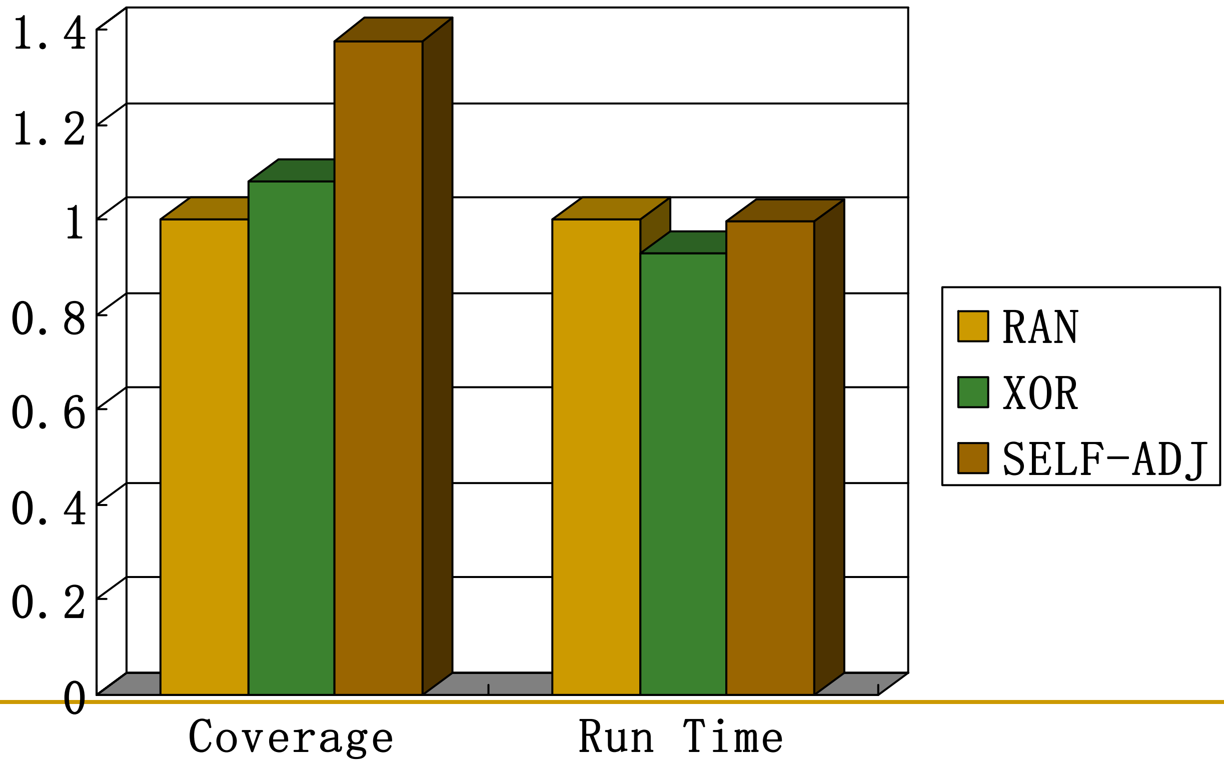


More Results

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

Analysis

- Average coverage and run time comparison
  - SELF-ADJ uses the same time to find the same number of test vectors with 37% higher coverage ratio than RAN



Conclusions and Future works

- Simplified Min-Distance-Sum is efficient and adequate for applications in constrained random stimulus generation
- When the test-case is difficult, the evaluation time can be ignored (Solving time >> Evaluation time)
- Self-adjusting method can improve the fault coverage ratio considerably
- Future works:
  - Algorithm optimization
  - Apply to high level functional verification

Thank You!