A Simulation-Based Test Generation Scheme Using Genetic Algorithms

M. Srinivas L.M. Patnaik
Microprocessor Applications Laboratory
Dept. of Computer Science and Automation
Indian Institute of Science, Bangalore 560 012, INDIA

Abstract

This paper discusses a Genetic Algorithm-based method of generating test vectors for detecting faults in combinational circuits. The GA-based approach combines the merits of two techniques that have been used previously for generating test vectors - the directed search approach and the random test method. We employ a variant of the traditional GA, the Adaptive GA (AGA), to improve the efficacy of the genetic search. Two cost functions that are used for assessing the quality of the vectors are discussed.

The performance of the AGA-based test generation approach has been evaluated using ISCAS-85 benchmark circuits. In our approach, the number of vectors that need to be simulated for detecting all detectable faults is significantly smaller than that required for a random test method. Even when optimized input distributions are used to generate the random test vectors, the AGA sustains its superior performance over the random test method.

1 Introduction

While it is a common practice to verify tests using simulation, several test generation schemes based on fault-simulation have been proposed in the literature (see for instance [5][6][17] [11][2][13][4]). In some of these techniques [5][6] fault simulation is employed in the fault propogation phase, although backtracking is still retained as an important component of test generation. Other techniques [11][17][13][2] [4] completely avoid backtracking, and rely solely on fault simulation to generate tests for detecting the faults.

Among test generation techniques that are based on fault simulation, directed search methods and random test methods form two distinct classes. Random test methods use input vectors that are derived from randomly generated 0s and 1s to detect the faults. The 0s and 1s of the random vectors are typically generated from circuit-specific optimized distributions. In contrast, the directed search methods generate each vector from previouly generated vectors in a deterministic fashion. The criterion for choosing a vector for fault-simulation is usually the minimization of a cost associated with the vector. The search proceeds along a direction of decreasing cost, and terminates at a stage when further reduction in cost becomes infeasible, i.e., the search terminates at a local optimum. The probability that the local optimum corresponds to a test vector [1] is high, though not equal to one.

In this paper we use Genetic Algorithms [8][9][10], robust search and optimization techniques, to generate tests more efficiently than random test methods. GAs achieve an optimal tradeoff between *exploitation* and *exploration*, the two distinctive features of directed search and random search respectively. Section 2 is devoted to an overview of GAs. Section 3 describes a variant of the traditional GA - the Adaptive GA (AGA)- that we have used in implementing the test generation system. To facilitate the search towards the test vectors, it is imperative to associate a cost with each vector, irrespective of whether it is a test vector for some fault or not. In Section 4, we discuss two cost functions that are used to evaluate how good the vectors are. Implementation details and simulation results are presented in Section 5. The paper is concluded in Section 6.

2 Overview of genetic algorithms

Genetic Algorithms are probabilistic search methods that employ a search technique based on ideas from natural genetics and evolutionary principles. They were conceived by Holland [10] in 1975, and since then, they have emerged as general purpose, robust optimization techniques (see [9], [10]).

Genetic Algorithms employ a random, yet directed search for locating the globally optimal solution. They are superior to gradient descent techniques as the search is not biased towards the locally optimal solution. On the other hand, they differ from random sampling algorithms due to their ability to direct the search towards relatively 'prospective' regions in the search space.

The GA may be viewed as an evolutionary process wherein the population of feasible solutions to the optimization problem evolves over a sequence of generations. During each *generation*, the fitness of each solution is evaluated, and solutions are *selected* for reproduction based on the relative fitness values.

Crossover causes the exchange of 'genes' between two randomly selected 'parents' to form new 'offspring'. The crossover occurs only with some probability (the crossover rate), and when the solutions

[1] We refer to a vector that detects atleast one undetected fault as a 'test vector'.

0-8186-3180-5/92 $3.00 © 1992 IEEE

are not subjected to crossover, they remain unmodified. *Mutation* involves the modification of the values of each 'gene' of a solution with some probability (the mutation rate).

3 Adaptive genetic algorithm

In the traditional GA, the crossver and mutation rates (p_c and p_m) are defined statically prior to the execution of the GA. In the AGA, p_c and p_m are determined for each solution in relation to its fitness value. Higher the fitness value, lower are the crossover and mutation rates. p_c and p_m are dynamic parameters and are adapted to each solution. p_c and p_m are varied as follows in the AGA :

$$p_c = k_1(f_{max} - f^{'})/(f_{max} - \overline{f}), \qquad f^{'} \geq \overline{f}, \qquad (1)$$

$$p_c = k_3, \qquad f^{'} < \overline{f}, \qquad (2)$$

and

$$p_m = k_2(f_{max} - f)/(f_{max} - \overline{f}), \qquad f \geq \overline{f}, \qquad (3)$$

$$p_m = k_4, \qquad f < \overline{f}, \qquad (4)$$

here
f is the fitness of a solution
$\overline{f}$ is the average fitness of the population
$f^{'}$ is the higher of the fitnesses of the two solutions undergoing crossover
f_{max} is the maximum fitness in the population

The parameters $k1$, $k2$, $k3$, and $k4$ are assigned the values 1.0 , 0.5 , 1.0 and 0.5 respectively. More details about AGA may be obtained in [14].

4 Cost functions

From the previous discussions in Sections 2 and 3 it is clear that the GA focusses its search towards solutions with higher fitness values, i.e., solutions with lower associated costs. An appropriate cost has to be associated with every vector irrespective of whether it detects any fault in the circuit or not.

We have experimented with two cost functions in our research. The first cost function , F1, is based on the distances the fault-effects of *activated* faults have been propogated in the circuit. The cost associated with each vector is given by the sum of the distances that the fault-effects of all the faults have propogated, and it decreases as the distances increase. This is given by :

$$C_v = \sum_{i \in F}(M - l_i) - d_i \qquad (5)$$

where
C_v : cost associated with vector v
M : maximum number of gate levels in the circuit
l_i : gate level at which the fault i is located
d_i : gate level to which the fault-effect of i has been propogated
F : the set of undetected faults

A drawback of F1 is that it does not account for the cost of activating a fault. Thus there is no selective pressure in the GA that leads it to generate vectors that can activate the faults. The activation of faults occurs purely by chance.

With the second cost function, F2, we aim at overcoming the drawback of F1. We associate with each vector a cost of activating a fault as well as a cost of propogating the fault effect. We use the idea of *active state* of a gate to determine this cost. Consider a gate that has inequiprobable 0s and 1s occurring at the output when equiprobable 0s and 1s are applied at the inputs. We define the state that occurs with a lower probability to be the *active state* of the gate. For instance the *active state* of an AND gate is 1, and for an OR gate it is 0. Intuitively, we may observe that it is easier to detect a fault that requires fewer gates to be in the active state (since we are considering a technique that uses random vectors). As an illustration, consider a fault at the output of a n-input AND gate. If the fault is a **sa0** fault, then the only vector that detects the fault is an all-1s vector, i.e., the vector that brings the gate into an active state. If the fault is a **sa1** fault, then any vector barring the all-1s vector will detect the fault. While the probability of detecting the **sa1** fault is close to 1.0, the probability of detecting the **sa0** fault is $\frac{1}{2^n}$. Similarly a **sa0** fault on any of the inputs can be detected only when the gate is in an active state. A **sa1** fault on a specific input i can be detected only if all the inputs excepting i are initialized to 1s and i is initialized to a 0. It is clear that a large percentage of the faults are detected when the associated gates are in an active state, or in a *near-active state*, i.e., when all inputs except one correspond to the vector that generates an active state. It may be noted that the remaining faults can be easily detected using random vectors. It is also important to note that the fault effect can be propogated through a gate only if it is in the active state. Based on the above ideas, F2 considers the fanin tree and the fanout tree of each fault and evaluates the number of gates that are in the active state in the two trees. The number of gates in the fanin tree of the fault site that are in the active state gives the activation cost of the fault. Similarly the number of gates in the active state in the fanout tree of the fault is the propogation cost of the fault. The cost of the vector decreases linearly with increasing number of gates in the active state as follows :

$$C_v = K - \sum_{i \in F}(a_{i_{in}} + a_{i_{out}}) \qquad (6)$$

where
C_v : the cost associated with the vector v
$a_{i_{in}}$: activation cost of fault i
$a_{i_{out}}$: propogation cost of fault i
K : a sufficiently large number to ensure that C_v is a positive

5 Experiments and results

Essential to the optimal performance of a GA are its control parameters : the population size , the mutation rate and the crossover rate. In the AGA, since we use dynamic crossover and mutation rates, we only have to make a choice of the population size. A population size of 100 has been found to be a good setting after performing preliminary experiments. We use the uniform crossover operator of [15].

Our experiments are based on the ISCAS-85 [3] benchmark circuits. For each circuit, the AGA has been run until all detectable faults have been detected. In the first set of simulations we have used equiprobable 0s and 1s to initialize the population. Also the mutations are generated with equiprobable 0s and 1s.

We have been able to detect all detectable faults in all the circuits except c2670 (we have not attempted c7552 due to restrictions on resources), which is very resistant to vectors generated from equiprobable 0s and 1s. Table 1 gives the number of vectors that have been fault-simulated before all faults have been detected. The results are tabulated for the two cost functions F1 and F2. Also indicated is the number of random vectors needed to be simulated for detecting all the faults [11]. The absolute fault coverage is also tabulated.

Circ.	Rand.	AGA-F2	AGA-F1	Cov.
c432	2048	500	1000	99.23
c499	4096	1100	1800	98.94
c880	16000	3700	5200	100.00
c1355	4096	3100	3400	99.49
c1908	8192	5700	7100	99.52
c3540	32000	7300	7600	96.00
c5315	8192	2100	3000	98.89
c6288	-	100	100	99.59

Table 1: Number of vectors simulated for detecting all detectable faults

It is clear that the AGA performs very well in comparison to a random test method. Also obvious is the superior performance of F2, clearly indicating that the inclusion of the cost of activating the faults has improved the performance of the AGA. For c2670 the AGA failed to show any significant improvement in the fault detection rate after the 85% mark.

In the second set of simulations we have used optimal input signal probabilities to generate the 0s and 1s of the vectors in the initial population of the AGA. We have obtained the optimal signal probabilities using the method proposed in [11]. The optimized signal probabilities have also been used to determine the distributions of 0s and 1s for the mutations. Table 2 gives the performance of the AGA with the cost function F2, when optimized signal probabilities are used. Also indicated is the number of pseudo-random vectors (also generated using the optimized signal probabilities) required for detecting all faults [11].

The data in Table 2 clearly indicates the superior performance of the AGA. For some circuits - c1355, c499, and c1908 - the number of vectors simulated in our method is almost 40-50% lesser than for Lisanke's method.

Ckt.	Random	AGA - F2
c432	320	300
c499	732	400
c880	160	200
c1355	2784	1500
c1908	3916	2100
c2670	6400	5100
c3540	4352	2800
c5315	1024	900
c6288	-	100

Table 2: Number of vectors simulated using optimized input signal probabilities

To evaluate the efficiency of the GA-based technique in terms of the consumed computing time, we have tabulated the computing overheads imposed by AGA and the signal probability evaluation. Table 3 gives the computation times of the different components, the execution being on a CD4360 UNIX machine. The columns of Table 3 give the total time, fault simulation time, time elapsed in computing the signal probabilities, the time consumed by AGA, and the percentage overhead in CPU time imposed by AGA and signal probability evaluation. For small circuits, the overheads are about 25%, but as the ciruit size increases, the overheads decreases to approximately 10 %. The overhead due to signal probability evaluation is the significant component, with the overhead due to the AGA being as low as 3% for the larger circuits. The trends indicated in Table 3 are very encouraging, the overheads are nominal, and steadily decrease as the circuit size increases.

Ckt.	Total	Sim.	Dis.	AGA	% OH
c432	4.467	3.567	0.550	0.350	25.23
c499	7.056	5.733	0.950	0.373	23.07
c880	6.917	5.533	0.883	0.501	25.01
c1355	76.933	70.167	4.367	2.399	9.60
c1908	96.201	88.210	5.474	2.510	9.05
c2670	758.330	694.584	41.250	22.496	9.17
c3540	543.900	497.500	33.117	13.283	9.21
c5315	190.567	176.067	10.283	4.217	8.2
c6288	36.483	33.533	0.250	2.700	8.79

Table 3: Computing overheads for GATES (all figures are in seconds)

6 Conclusions

Genetic Algorithms are being used successfully in a variety of problem domains : Structure optimization, Pipeline optimization VLSI Cell placement, etc. In this paper we have discussed the application of GAs to the task of generating test vectors for faults occuring in combinational logic circuits. The motivation for this work has been to improve upon the performance of random test methods by incorporating a directed search mechanism to locate the vectors, thereby exploiting the information contained in the previously generated vectors.

The results from our simulations on ISCAS-85 benchmark circuits indicate that our approach is significantly superior to any previous random test method. We have been able to reduce the number of test vectors to be simulated significantly : almost 50% for some ISCAS-85 benchmark circuits. Moreover the Genetic search can be integrated into any random test method, leading to an improvement in the performance of the random test method. The overheads in computation time due to AGA and signal probability evaluation are as low as 8 % for large circuits, and decrease asymptotically as the circuit size increases. Test generation for faults in sequential digital circuits is a significantly harder problem than for combinational circuits, due to the presence of stored internal states. Extending this work to encompass sequential circuits demands some tricky issues to be tackled : (i) how are costs to be associated with the storage elements? (ii) how are vectors to be generated in relation to the current state of the circuit? We are looking at these problems and are trying to develop a general GA based test generation system that can handle sequential and combinational circuits.

Acknowledgements

We thank Dr. V.D. Agrawal for his useful suggestions in the early stages of this work.

References

[1] K. T. Cheng and V.D Agrawal, *Unified Methods of VLSI Simulation and Test Generation*, Kluwer Academic Publishers, 1989.

[2] K. T. Cheng, V. D. Agrawal, and E.S. Kuh, ' A Sequential Circuit Test Generator Using Threshold Value Simulation', Fault Tolerant Computing Symp. (FTCS - 19), June 1988, pp24-29.

[3] 'Special Session: Recent Algorithms for Gate-Level ATPG with Fault Simulation and their Performance Assessment", Proc. of the 1985 IEEE Int. Symp. Circuits and Systems (ISCAS), June 1985, pp. 663-698.

[4] V. D. Agrawal, K.T. Cheng and P. Agrawal, 'A Directed Search Method for Test Generation Using a Concurrent Simulator', IEEE Trans. on CAD, Vol. 8 , Feb. 1989, pp. 131-138.

[5] T. J. Snethen, 'Simulator Oriented Fault Test Generator', Proc. 14th Design Automation Conference, June 1977, pp. 88-93.

[6] Y. Takamatsu and K. Kinoshita, 'CONT : A Concurrent Test Generation Algorithm' , Fault Tolerant Computing Symp. (FTCS - 17), June 1979, pp. 180-184.

[7] P. Agrawal , V. Agrawal, 'Probabilistic Analysis of Random Test Generation Method for Irredundant Combinational Logic Circuits', IEEE Transactions on Computers, Vol. C-24, July 1975, pp. 691 - 695.

[8] L. Davis *Handbook of Genetic Algorithms*, Van Nostrand Reinhold Publishers, 1991.

[9] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, 1989.

[10] J. H. Holland, *Adaptation in Natural and Artificial Systems*, Ann Arbor : University of Michigan Press, 1975.

[11] Lisanke et al., 'Testability driven Random Test-Pattern Generator', IEEE Transactions on CAD, Vol. CAD-6, Nov. 1987, pp. 1082-1087.

[12] J. D. Schaffer et al., 'A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization' Proc. of the Third International Conf. Of Genetic Algorithms, 1989, pp. 51-60.

[13] H. D. Schurmann, 'The Weighted Random Pattern Test Generator', IEEE Transactions on Computers, Vol. C-24, July 1975, pp. 695-700.

[14] M. Srinivas and L. M. Patnaik, 'Adaptive probabilities of Crossover and Mutation in Genetic Algorithms', Technical Report, Tr. No. MAL-9102, Microprocessor Applications Laboratory, Indian Institute of Science.

[15] G. Syswerda, 'Uniform Crossover in Genetic Algorithms', in Proc. of the Third International Conf. on Genetic Algorithms, 1989, pp. 2-9.

[16] H. J. Wunderlich, 'PROTEST : A Tool for Probabilistic Testing Analysis', Proc. of the ACM IEEE 22nd DAC, 1985, pp. 204-211.

[17] H. J. Wunderlich, 'Multiple Distributions for Biased Test Patterns', IEEE Transactions on Computers, Vol. C-9, June. 1990, pp. 584-593.