Level-Oriented GA-Based Test Generation of Logic Circuits

Wangning Long, Shiyuan Yang, Yinghua Min *, Shibai Tong

Automation Department
Tsinghua University
Beijing, 100084, P.R.China

* CAD Lab., ICT
The Chinese Academy of Sciences
Beijing, 100080, P.R.China

Email: longwn@hotmail.com, ysy-dau@mail.tsinghua.edu.cn, min@mimi.cnc.ac.cn

Abstract - **This paper investigates the technique of level-oriented GA-based test generation of logic circuits. The fitness calculation method is improved. Experimental results show that the improved fitness calculation method is more effective than the previous one. Besides, it is also showed that GA method is better than normal random method, and the adaptive P_c [1] does not necessarily result in higher test generation performance. With the rapid development of parallel computing, GA-based test generation is promising for practical applications.**

I. INTRODUCTION

Genetic algorithm (GA) is a kind of guided random method. It imitates the mechanism of genetic evolution of creatures. GA is effective and robust in solving optimization problems. Besides, GA is convenient for parallel computing. Therefore it is widely used in many fields nowadays.

The problem of test generation is NP completed. Although some test generation systems have been reported, the problem remains a hot topic. This is mainly because of the fact that circuits are becoming larger and larger.

Recently, some researchers have tried to solve the test generation problem by using genetic algorithm[1-3]. M. Srinivas and L. M. Patnaik present an adaptive crossover and mutation probability calculation method in [1], based on which adaptive GA (AGA) is proposed. They make some attempts to apply AGA to test generation of stuck-at fault for combinational circuits. They choose circuit level where a fault signal is propagated as the basis of calculating fitness. It is claimed in [1] that the results are obviously superior to that of the simple GA (SGA). However, the experiments and analysis in [1] are limited.

In this paper, some detailed investigation and analysis of level-oriented GA-based test generation of logic circuits are presented. Meanwhile the fitness calculation method is improved to meet the requirements of multi-output circuits. Experimental results show that our method is more effective than the previous one for most ISCAS'85 benchmark circuits.

The remainder of this paper is organized as follows. In section II, the approach and the algorithm of the level-oriented GA-based test generation is discussed in detail and improvement of fitness calculation is made. In section III, some experimental results are given. Finally in section IV, conclusion is made.

II. APPROACH AND ALGORITHM

First, some important concepts of the GA-based test generate of combinational circuits are introduced briefly.

An individual, or a chromosome, is a binary vector which represents a primary input pattern of a circuit. The length of an individual equals to the number of the primary inputs of the circuit under test.

A generation is a group of individuals. The number of individuals in a generation is usually a constant, which is determined by the complexity of the circuit. The first generation is usually produced by a random method, and the following generation is produced by using selection, crossover, and mutation, which are taken as three operators of genetic algorithms.

How to select individuals from the generation is important to GA applications. Roulette wheel is a common used selection method, by which the better the individual is, the higher the choosing probability is. The goodness or badness of an individual is determined by its fitness, which is an re-

0-7803-4253-4/97/$10.00 © 1997 IEEE

search emphasis in most GA applications. A suitable fitness calculation method will make GA more effective; otherwise, GA may be inefficient, even worse than normal random method. When an optimization problem is maximization problem, the higher is an individual's fitness, the better is it, and the higher should be the probability of choosing it.

After two individuals are selected, crossover is performed according to the crossover probability (P_c). First a uniform random number x ($0 \le x \le 1$) is generated. If $x \le P_c$, the two vectors exchange some bits with each other so as to generate two new individuals; otherwise, the two individuals are unchanged. Crossover is an important feature of GA method. With crossover operation, the goodness of the parents may be combined so as to make the new individuals better than their parents. In our system, single-point (middle point) binary crossover is adopted.

Mutation is another important feature of GA. For every bit of an individual, a uniform random number x ($0 \le x \le 1$) is generated. If $x \le P_m$ (probability of mutation), the bit is set to the opposite value, *i.e.*, $1 \rightarrow 0$, or $0 \rightarrow 1$. Mutation introduces some changes into the new generation so as to prevent the premature convergence of the population.

Figure 1 shows the main frame of the GA-based test generation algorithm. The first generation is generated by random method. To evaluate the individuals in the current generation, fault simulation is carried out for each one. The fault simulator used in our system is the concurrent fault simulator[4]. Once a fault is detected by an individual, it is deleted from the fault set. Then the convergence condition (stop condition) is checked. If the convergence condition is not achieved, some pairs of individuals will be selected according to their fitness to do crossover and mutation. By this way a new generation is produced. The procedure is continued until the convergence condition is achieved and then the final report is printed.

```
GA-based Test Generation( )
{
    Initialize the first population;
    do {
        for (every individual x in the current population) {
            Take fault simulation with x as the PI vector;
            Evaluate the fitness of x;
        }
        if (convergence condition is not achieved)
            for ( i = 0;  i < popsize / 2;  i++ ) {
                Select a pair of individuals;
                Do crossover and mutation;
            }
    } while (convergence condition is not achieved);
    Make final report;
}
```

Figure 1 Main frame of GA-Based Test Generation

What information is taken as the basis of calculating fitness are fatal to GA applications. In [1], the level where a fault signal is propagated and the maximum level of the circuit are taken as the basis of calculating fitness. The formula of calculating fitness in [1] is as follows:

$$f_j = \sum_{i \in F} (l_m - l_{ji}) \quad (1)$$

where f_j ($j = 1, 2, \cdots, N$) is the fitness of the jth individual, N is the number of the individuals in a generation, l_m is the maximum level of the circuit, l_{ji} is the level where the signal of fault i is propagated when the jth individual is taken as the primary input vector, and F is the undetected fault set. The objective of the genetic algorithm is to minimize f_j.

Although (1) reflects the fact that the smaller the f_j is, the better the individual seems, it is not true in most practical circuits which are usually multi-output circuits and their primary outputs locate at different levels. Once a fault signal is propagated to any primary output, the fault is regarded as detected. PO_1 and PO_2 in figure 2, for example, do not locate at the same level. Without loss of generality, suppose a stuck-at fault appear at primary input PI_2. If individual $\boldsymbol{a}$ propagates the fault signal to point A which locates at the 2nd level, and another individual $\boldsymbol{b}$ propagates it to point B which locates at the 3rd level. The maximum level of the circuit is 5. According to (1), vector $\boldsymbol{b}$ is superior to vector $\boldsymbol{a}$. However, in fact, as point A is only one level from the primary output PO_1 and point B is two level from the primary output PO_2, it is obvious that vector $\boldsymbol{a}$ is superior to vector $\boldsymbol{b}$.

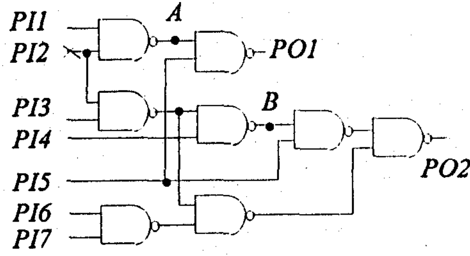

Figure 2 An example circuit whose primary outputs locate at different levels

It is noticed that to adopt the maximum level of the circuit as a parameter in (1) is not suitable. Instead, the level of the nearest primary output from the location where the fault signal is propagated is a better choice to evaluate a individual. Therefore, (1) is improved as following:

$$f_j = \sum_{i \in F}(l_{oji} - l_{ji}) \tag{2}$$

where l_{ji} is the level where the signal of the ith fault can be propagated when the jth vector is taken as the primary input pattern, l_{oji} is the level of the nearest primary output from the propagated point, and the rest parameters are similar to that in (1). In the next section, it will be showed by experiments that (2) is more effective than (1).

Srinivas and Patnaik propose to adopt adaptive P_c and adaptive P_m in [1], because they thought that AGA is more effective than SGA. But the fact is that although the adaptive P_c is effective in many GA applications, it seems unnecessary in case of test generation. If P_c is less than 1, there is some chance that some individuals in a generation may exist in the next generation, but no faults can be detected when they are used to do fault simulation. The only effect is that it can be taken as seeds which is passed to the next generation. Therefore it may sometimes be more effective to set P_c to 1. However, it is still a good idea to use adaptive P_m, which makes it possible to introduce more changes to a worse individual and avoid too much changes for a better individual.

In [1], P_c is calculated by (3) and (4) and P_m is calculated by (5) and (6) for maximization problems as follows:

$$p_c = k_1(f_{max} - f')/(f_{max} - \bar{f}) \quad \text{, if } f' \geq \bar{f} \tag{3}$$

$$p_c = k_3 \quad \text{, if } f' < \bar{f} \tag{4}$$

$$p_m = k_2(f_{max} - f)/(f_{max} - \bar{f}) \quad \text{, if } f \geq \bar{f} \tag{5}$$

$$p_m = k_4 \quad \text{, if } f < \bar{f} \tag{6}$$

where k_1, k_2, k_3 and k_4 are constant, which we set as $k_1 = k_2 = 1$, and $k_3 = k_4 = 0.5$, f_{max} is the maximum fitness in the current generation, $\bar{f}$ is the average fitness of the individuals in the generation, f' is the larger fitness of the two individuals selected, and f is the fitness of the individual that will do mutation.

For minimization problems, P_c is calculated by (7) and (8) and P_m is calculated by (9) and (10) as follows:

$$p_c = k_1(f' - f_{min})/(\bar{f} - f_{min}) \quad \text{, if } f' \leq \bar{f} \tag{7}$$

$$p_c = k_3 \quad \text{, if } f' > \bar{f} \tag{8}$$

$$p_m = k_2(f - f_{min})/(\bar{f} - f_{min}) \quad \text{, if } f \leq \bar{f} \tag{9}$$

$$p_m = k_4 \quad \text{, if } f > \bar{f} \tag{10}$$

where f_{min} is the minimum fitness in the current generation, f' is the less fitness of the two individuals selected, and the rest parameters are similar to the above.

In this paper, the calculation of P_m adopts (5)(6) or (9)(10), while 'P_c with adaptive' means that P_c is calculated by (3)(4) or (7)(8) and 'P_c without adaptive' means that P_c is set to 1.

III. EXPERIMENTAL RESULTS

The level-oriented GA test generation system is implemented at the sun 4/20 workstation. The ISCAS'85 benchmark circuits are used to scale the test performance. The concurrent fault simulator[4] are used to take fault simulation. The fault set is the unreduced stuck-at fault set. A common used index in GA application, the number of generation, is adopted to evaluate the performance of various method, including random method. The number of individuals in a generation is set to 100. For example, 16 generation means that 1600 vectors are generated in the whole process, and the concurrent fault simulation is called for 1600 times.

The comparison between GA and random method is listed in table 1, in which 'No.G' denotes the number of generation, 'F.C.' denotes the fault coverage. The convergence condition is set as following: the program stops when (i) fault coverage > 0.99, or (ii) there are continuous 6 generations which have the same fault coverage.

In table 1, GA2 uses the improved fitness calculation formula (2). The objective here is to minimize f_j, which means

Table 1 Comparison of random method and GA method

Circuit Name	Random		GA2	
	No.G	F.C.	No. G	F.C.
C432	13	0.9624	9	0.9624
C499	11	0.9909	12	0.9909
C880	31	0.9915	24	0.9896
C1355	13	0.9915	13	0.9915
C1908	31	0.9900	39	0.9914
C2670	20	0.8352	16	0.8350
C3540	32	0.9606	26	0.9604
C5315	8	0.9904	6	0.9943
C6288	1	0.9943	1	0.9943
C7552	24	0.9367	21	0.9372

that the smaller is f_j, the higher is the probability of choosing it. Therefore, the formula of calculating selection probability P_j is devised as following:

$$P_j = (1 - \frac{f_j}{\sum_{i \in F} f_i}) \frac{1}{|F|-1} \quad (11)$$

where $|F|$ denotes the size of the undetected fault set F.

Table 1 shows that for 10 benchmark circuits, there are 6 circuits for which GA2 is better than random method, while only 2 circuits for which GA2 is worse than random method. It shows that C2670 is hard-to-test by both GA2 and random method. Besides it also shows that for C6288, both methods reach a fault coverage as high as 99.43% at the end of first generation. Therefore C6288 is ease-to-test by random method, and GA's effectiveness can not be noticed in this circuit.

Figure 3 lists the comparison of the average performance between GA and normal random method. GA2 is also used here. The curves in the figure 3 are the average performance of testing the 10 benchmark circuits of ISCAS'85 with the two methods. The X-axis denotes the number of generation, and the Y-axis denotes the average fault coverage. It shows that the average performance of GA2 is better than that of random method from the 10 to 18 generation.

Table 2 lists the comparison of various fitness calculation methods. GA1 adopts the fitness calculation method in [1]

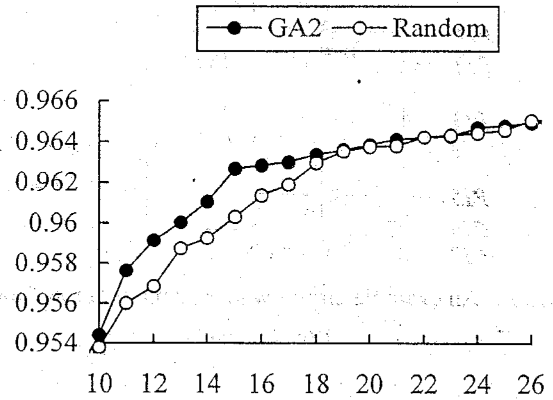

Figure 3 Comparison of average performance of GA and random method

which employs fitness calculation formula (1) and P_j calculation formula (11). GA2 is the same as in table 1. GA3 is similar to the GA2 except that its fitness f_j is set to another form so that the problem is become to a maximization problem. The calculation of f_j and P_j is as follows:

$$f_j = \frac{\sum_{i \in F} (l_m - (l_{oji} - l_{ji}))}{|F| \times l_m} \quad (12)$$

$$P_j = \frac{f_j}{\sum_{i \in F} f_i} \quad (13)$$

Table 2 The comparison of various fitness calculation method

Circuit Name	GA1		GA2		GA3	
	No.G	F.C	No.G	F.C	No.G	F.C.
C432	7	0.9624	9	0.9624	5	0.9616
C499	13	0.9922	12	0.9909	11	0.9910
C880	20	0.9761	24	0.9869	31	0.9778
C1355	11	0.9908	13	0.9918	11	0.9908
C1908	30	0.9903	39	0.9914	30	0.9911
C2670	17	0.8354	16	0.8350	11	0.8352
C3540	46	0.9616	26	0.9604	27	0.9600
C5315	9	0.9902	6	0.9910	6	0.9911
C7552	34	0.9357	21	0.9372	23	0.9370

All the methods in table 2 are with adaptive P_c and P_m so as to compare them conveniently. The convergence condition is the same as in table 1.

Table 2 shows that for 9 benchmark circuits, GA2 is better than GA1 for 5 circuits. Besides, GA3 is better than GA1 for 6 circuits, and is equal to GA1 for two of the other 3 circuits. It is obvious from table 2 that the improved fitness calculation method is more effective than previous one in guiding the individual selection.

Table 3 lists the test results of the GA methods with and without adaptive P_c. GA1 and GA3 have been discussed in table 2, which adopt adaptive P_c and P_m. GA4 is similar to GA3 except that P_c=1, and GA5 is similar to GA4 except that $p_m = k_2[(f_{max} - f)/(f_{max} - \bar{f})]^2$ when $f \geq \bar{f}$.

Table 3 Comparison of cases with and without adaptive P_c

Circuit	GA1		GA3		GA4		GA5	
Name	No.G	F.C	No.G	F.C	No.G	F.C	No.G	F.C.
C432	7	0.9624	5	0.9616	5	0.9608	5	0.9608
C499	13	0.9922	11	0.9910	11	0.9922	10	0.9912
C880	20	0.9761	31	0.9778	26	0.9920	18	0.9919
C1355	11	0.9908	11	0.9908	16	0.9908	13	0.9900
C1908	30	0.9903	30	0.9911	26	0.9906	31	0.9900
C2670	17	0.8354	11	0.8352	15	0.8365	15	0.8365
C3540	46	0.9616	27	0.9600	30	0.9606	27	0.9600
C5315	9	0.9902	6	0.9911	13	0.9904	8	0.9905
C7552	34	0.9357	23	0.9370	24	0.9376	44	0.9360

Table 3 shows that there is no obvious difference among GA3, GA4 and GA5 concerning with their test performance, while GA1 is worse than GA4 and GA5 in most cases. The obvious difference in test performance between the AGA and SGA in [1] is mainly because their SGA adopted improper P_c and P_m, which is P_c = 0.68 and P_m = 0.005. Table 3 illustrates that adaptive P_c do not necessarily improve the test performance. Usually, P_c could be just set to 1.

IV. CONCLUSION

Above analysis and experimental results illustrate that as a kind of guided random method, GA is superior to the normal random method concerning with the test performance. But it is still a kind of random method, and it will be difficult to test the rest hard-to-test faults. It is better to employ other methods, such as deterministic test pattern generation, to test the rest faults.

It also shows that our improvement of level-oriented GA-based test generation method, which adopts the distance between the level to which the fault signal can be propagated and the level of nearest primary output as the basis of fitness calculation, is more effective than the previous one.

Lastly, the experimental results show that although it is a good ideal to use adaptive P_m, adaptive P_c does not necessarily result in higher test performance as claimed in [1]. In some cases P_c = 1 seems to be more effective.

As GA is suitable for parallel computation, GA test generation method is promising to be practical in the near future with the rapid development of the parallel computation technique.

V. ACKNOWLEDGMENT

This work was supported by the CAD lab. of the Chinese Academy of Science. The authors thank Prof. Zongcheng Li for his useful discussion and providing the fault simulator.

VI. REFERENCES

[1] M. Srinivas, L. M. Patnaik, "Adaptive Probabilities of Crossover and Mutation in Genetic Algorithm", IEEE Trans. System, Man and Cybernetics, Vol.24, No.4, April, 1994, pp. 656-667.

[2] Saab, Y. G. Saab, J. Abraham, "CRIS: A Test Cultivation Program for Sequential VLSI Circuits", In: Proc. ICCAD-90, 1992, pp. 216-219.

[3] Hayashi, H. Kita, K. Hatayama, "A Genetic Approach to Test Generation for Logic Circuits", In: Proc. of 3rd Asian Test Symposium, 1994, pp. 101-106.

[4] Ulrich, T.Baker, "The Concurrent Simulation of Nearly Identical Digital Network", In: Proc. 10th Design Automation Workshop, Vol.6, 1973, pp. 145-150.

[5] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Publishing Company, USA, 1989.