Forecasting the Efficiency of Test Generation Algorithms for Digital Circuits[*]

Shiyi Xu and Wei Cen

School of Computer Science & Engineering
Shanghai University, Shanghai, China 200072
Email: syxu@fudan.ac.cn

Abstract – *Within this era of VLSI circuits, testability is truly a very crucial issue. To generate a test set for a given circuit (including both combinational and sequential circuits), choice of an algorithm within a number of existing test generation algorithms to apply is bound to vary from circuit to circuit. In this paper, the genetic algorithms are used to construct the models of existing test generation algorithms in making such choice more easily. Therefore, we may forecast the testability parameters of a circuit before using the real test generation algorithm. The results also can be used to evaluate the efficiency of the existing test generation algorithms. Experimental results are given to convince the readers of the truth and the usefulness of this approach.*

Index Terms – Testability, Genetic Algorithm, Forecasting, Test Generation.

I. INTRODUCTION

Test generation algorithms are being developed increasingly with the continuous creations of incredible sophisticated computing systems. For the time being, a number of techniques, test generation algorithms, for testing of stuck-at-fault and designing of testable systems are in existence [1]. For logical stuck-at faults, such well-known algorithms including D [2-5], PODEM (Path-Oriented Decision Making) [6], FAN (Fanout-Oriented TG) [7] , 9-V [8,9] and other algorithms for a given combinational circuit and such well-known algorithms including "Transitive Closure algorithm" [27] for a given sequential circuit are being widely used in practice to generate the test patterns under test (CUT); TOPS (Topological Search) [10], FAST (Fault-Oriented Algorithm for Sensitized-path Testing) [11], RTG (Random Test Generator) and others [4] are also available in some special cases. However, for a given CUT (including both combinational and sequential circuits), the choice of which existing algorithm to use is not always a straightforward issue. This is because different test generation algorithms are bound to perform differently when applied on the same CUT. In other words, it is not always true that one algorithm will be suitable for all different circuits being tested. Instead, the different algorithms will perform better in different circuits. We use the term *efficiency* to measure the goodness of a given algorithm for a given CUT. By high efficiency of an algorithm, we mean that (1) there should be less *test patterns* (TP) produced by the algorithm than by others, (2) the higher *fault coverage* (FC) should obtain by using these test patterns and (3) should have relatively short *CPU time* (TCPU) on which the algorithm will spend when running. Hence, the above three parameters TP, FC and TCPU are called *testability parameters*, which are the important, and/or essential profiles/factors that determine the test pattern generation cost for testing of circuits. Therefore, a test generation algorithm of high efficiency implies that it produces fewer test patterns while covering more faults in the CUT with less CPU time than other algorithms.

With the increasing technology advancements, usually the parameter TCPU is going to be diminishing in the background. For this paper, therefore, only TP and FC are to be considered.

Unfortunately, the determination of efficiency of an algorithm concerned may not be a straightforward job. This is just because a prior knowledge on the performance of these algorithms for a particular circuit is usually not precisely available in making such a choice. Hence, in doing so, what we need is to know the exact testability parameters – TP and FC[12-14] – for a particular circuit before a designated algorithm is applied. One practical way of obtaining these Forecasted Values testability parameters directly without need of executing the real algorithms concerned is to establish a model for each algorithm mentioned above so that the analysis of efficiency of an algorithm may become simple and precise. The models can be constructed in variety of fashions as long as the model established is able to emulate the algorithm precisely. As a matter of fact, much work has been done in this area. For example, a statistical model [12] for forecasting the testability parameters of algorithms for sequential circuits is in existence much as Neural Network models [13,14] for both sequential and combinational circuits exist. Since the problem of locating testability parameters is more of nonlinear in nature, we develop a Genetic Algorithm Forecasting Model (GAFM) in this paper using some ideas given in the relevant materials [15-17] to model some well-known test pattern generation algorithms. Moreover, consideration for various GA-related issues [18-21] is taken.

In this paper, the GAFM is developed for each test generation by a genetic algorithm approach and implemented based on the benchmark circuits [13] giving forecasts for the testability parameters of the algorithms considered. The models presented in this paper are then applied on some (commercial) combinational and sequential circuits for demonstration purposes.

II. DEFINING CHARACTERISTICS AND TESTABILITY PARAMETERS

Since their introduction, the benchmark circuits have been widely distributed and also used not only for the purpose of

[*] This work was supported by National Natural Science Foundation of China (NSFC) under the grant No. 69873030

evaluating the performance of Automatic Test Pattern Generation (ATPG) algorithms, but also in areas of simulation, testability analysis, formal verification, logic synthesis, technology mapping and layout synthesis. Here, we will establish a testability model for each of well-known test generation algorithms. However, some characteristics of a circuit need to be defined first before testability models can be established. The following definition states their properties.

Definition 2.1: The *Defining Characteristics* (listed below) of a combinational or a sequential circuit are the important and/or essential structural characteristics that can be used to identify the circuit. We adopt a three-letter notation for the defining characteristics outlined below.

For a combinational circuit, we have 7 defining characteristics parameters:

TNL: Total Number of Lines in the circuit
NPI: Number of Primary Inputs
NPO: Number of Primary Outputs
FOS: Number of Fanout Stems
FOB: Number of Fanout Branches
INL: Internal Number of Lines:
$= TNL - (NPI + NPO + FOB)$
NIL: Net Internal number of Lines: $= INL - FOS$

The last two are calculated fields. Hence, the end user specifies only the first five.

In their CDL (Circuit Description Language) format, the circuits' defining characteristics can be extracted. We use this approach on the benchmark circuits and compile the table 2.1 below.

Tab 2.1: Defining Characteristics for the benchmark circuits

Circuit	***TNL***	***NPI***	***NPO***	***FOS***	***FOB***	***INL***	***NIL***
C17	17	5	2	3	6	4	1
C432	432	36	7	89	236	153	64
C499	499	41	32	59	256	170	111
C880	880	60	26	125	437	357	232
C1355	1355	41	32	259	768	514	255
C1908	1908	33	25	385	995	855	470
C2670	2670	233	140	454	1244	1053	599
C3540	3540	50	22	579	1821	1647	1068
C5315	5315	178	123	806	2830	2184	1378
C6288	6288	32	32	1456	3840	2384	928
C7552	7552	207	108	1300	3833	3404	2104

Table 2.2 shows the results of this procedure for the 10 non-benchmark (commercial) circuits.

Tab 2.2: Defining Characteristics for the non-benchmarks

Circuit NAMES (54/74 Family)	***TNL***	***NPI***	***NPO***	***FOS***	***FOB***	***INL***	***NIL***
SN54LS157/	37	10	4	4	12	11	7
SN54157	42	10	4	3	17	11	8
SN54182	70	9	5	9	42	14	5
SN5444A	70	4	10	8	48	8	0
SN5483A	104	9	5	17	59	31	14
SN54147	84	9	4	11	44	27	16
SN54148	95	9	5	18	57	24	6
SN54155	48	6	8	6	27	7	1
SN54181	192	14	8	23	112	58	35
SN5449	93	5	7	9	54	27	18

For a sequential circuit, we have 16 defining characteristics:

1. NPI: Number of primary inputs
2. NPO: Number of primary output
3. NOG: Number of gates
4. NFF: Number of flip-flops
5. SED: Sequential Depth

Each node in the sequential circuit is defined as the minimum number of flip-flops on the path to any primary output of the circuit. The **sequential depth (SD)** of the whole circuit is then defined as the maximum sequential level of its nodes.

6. NOL: Number of synchronous loops

Due to the presence of fanout points (FP) in the circuits, the exact count of the loops may vary from programmer to programmer. Researcher gives the pseudo code of the algorithm used in our loop identification.

7. FFL: Number of flip-flop per loop
8. LFF: Number of loop per flip-flop
9. FOS: Number of fan-out stems
10. PFB: Number of prime fan-out branches

In a sequential circuit, a fan-out branch is said to be prime if and only if all the propagation paths from other fan-out branches of its fan-out stem to the primary outputs pass through this fan-out branch.

11. NSG: Number of non-SAD gates

A gate in a sequential circuit is said to be non-SAD, if the gate is self-hiddening but not delay-reconvergence [13]. The number of non-SAD gates in a circuit will have a great influence on testability, therefore, it should be selected as one of the characteristic parameters.

12. NSF: Number of non-SAD flip-flops

In a sequential circuit, a D flip-flop is said to be non-SAD, if the D flip-flop is on paths which pass through different numbers of delay elements and have different inversion parities and reconverge.

13. NOF: Number of non-observable flip-flops
14. NOG: Number of non-observable gates
15. FLT: Total number of faults
16. ISC: Primary inputs which the circuit state directly controlled

III. Genetic Algorithm Forecasting Model (GAFM)

Any abstract task to be accomplished can be thought of as solving a problem, which, in turn, can be perceived as a search through a space of potential solutions. Since one is after "the best" solution, this task can be viewed as an optimization process. For small spaces, classical exhaustive methods usually suffice; for larger spaces special artificial intelligence techniques must be employed. *Genetic Algorithms* (GAs) are among such special techniques; they are stochastic algorithms whose search methods model some natural phenomena: genetic inheritance and Darwinian strife for survival. [16,18-21] Genetic algorithms are a class of general-purpose (domain independent) search methods that strike a remarkable balance between exploration and exploitation of the search space.

Crossover combines the features of two parent chromosomes to form two similar offspring by swapping corresponding segments of the parents. The intuition behind its applicability is information exchange between different potential solutions. On the other hand, the intuition behind the mutation operator is the introduction of some extra variability into the population. *Mutation* arbitrarily alters one or more genes of a selected chromosome, by a random change with a probability equal to the mutation rate. Many evolution programs can be formulated for a given problem. Such programs may differ in many ways. These ways include use of different data structures for implementing a single individual, "genetic" operators for transforming individuals, methods for creating an initial population, methods for handling constraints of the problem, and parameters (such as population size, probabilities of applying different operators, and maximum number of generations).

Before we introduce our solution, let us first give some definitions that will be used later.

- **Individual**: In genetic algorithms, an individual is the representation of a potential solution to a given problem.
- **Sample space**: All individuals constitute a sample space. Evaluating the solution to a question is searching an optimum solution in the sample space. Genetic algorithms can help to accelerate finding the solution by giving some "instructions".
- **Genes**: A function hereditary unit that occupies a fixed location on a individual, has a specific influence on phenotype, and is capable of mutation to various allelic forms.

3.1 REPRESENTATION FOR INDIVIDUALS:

Here, in this case, we define each of individuals as an expression. An expression can be defined according to the following regulations:

1. Any characteristic parameter or constant number can be an expression.
2. The composite of expressions by mathematical operations constitutes an expression. Expressions can be obtained by using the above two rules repeatedly. Followings are some examples of individuals:

- (th(FOS))+(sin((NOF)-(NFF)))
- ((((NSG)+(4.858))+(sqrt(NSG)))/((0.956)/(3.563)))/(((SD)+(NOFF))+(sqrt(NOG)))
- (((((cuberoot(NSG))/((PO)/(TF)))-(sqrt(FF)))-((SAPI)*(LFF)))*(SAPI))/((((((((((1.661)+(quint (NSFF)))- (PFB))*((NOFF) +(quint(NSFF))))*(LFF))/(NSG))*(LFF))-(LFF))+(8.491))*(NOFF))+(sqrt(FF)))

A gigantic space constructed by all the expressions is called a sample space. It is necessary to point out that the combinational circuits and sequential circuits have different structures and characteristic parameters, so the spaces of combinational and the sequential circuits expressions can not be confused each other. Forecasting testability parameters of test generation algorithms will be based on these two spaces according to the type of the circuits the algorithm serves.

For simplicity, each expression can be represented by a tree. For example, the first expression mentioned above can be expressed as shown in Fig. 3.1.

The leaf nodes in our representation schema can be:

- one of the 7 (for combinational circuits) or 16 (for sequential circuits) characteristic parameters of the circuit, or
- a constant number, which varies from 0.001 to 9.999

The interior node represents mathematical operator have been defined as one of the following five types of the operators:

- basic mathematical operators: +, -, *, /
- trigonometric functions and antitrigonometric functions: sin(x), cos(x), arctg(x)
- logarithmic functions: log(x), ln(x)
- exponential functions: e^x, 10^x
- hyperbola functions: sh(x), ch(x), th(x)
- power functions: x^2, $x^{0.5}$, x^3, $x^{1/3}$, x^5, $x^{0.2}$,x^{-1}

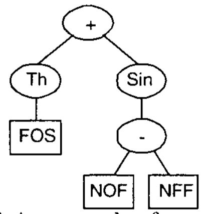

Fig. 3.1 An example of tree-type expression

3.2 INITIAL POPULATION:

Individuals at the 1st population are randomly generated by the system. When generating initial population, the system randomly chooses the above functions, constant numbers, and the characteristic parameters and then combines them into a complete expression.

To prevent system from generating trees that are too complex to run, some restrictions on the maximum number of the operators in an expression have been set.

The system employs a "top-down" method when generating a random expression. In this method, the root of the expression tree is first generated, then the second level of the tree is generated. Then generates the third, fourth, fifth,... level of the tree. Whenever a desired number of operators is generated, the system generates the leaf nodes according to the number that these operators needed. Finally, the generated trees are converted into the prefix representation of the expression tree. Each time when a new expression is generated, it was first compared with the existing individuals. If the individual is identical to some individual previously generated in the population, it will be rejected.

3.3 FITNESS EVALUATION

Once initial individuals are randomly generated according to the way mentioned above we then have to optimize the individuals to obtain the best results. Therefore, a numerical fitness function is needed in order to quantitatively evaluate the suitability of each individual.

Let *e(i,j)* be the Forecasted Values testability parameter of j^{th} circuit for i^{th} individual, and *a(j)* the actual testability parameter of j^{th} circuit. Assume that there be *C* circuits. Then we define

$$f(i) = \sum_{j=1}^{C} | e(i,j) - a(j) |$$

It can be seen that $f(i) \geq 0$ and should be the less the better for this individual.

3.4 GENETIC OPERATIONS

The crossover and mutation processes are the most important parts of the GA. The performance is basically done by these two operators.

As will be seen, the crossover operation is well defined and syntactically legal for any two evaluating expressions and any two crossover points. For example, consider the parental individuals in Fig 3.2. The functions in these individuals are + , - , * , / . The terminals are 0.1 , 10 , PI , SAPI , TF , NOG , L , PFB .

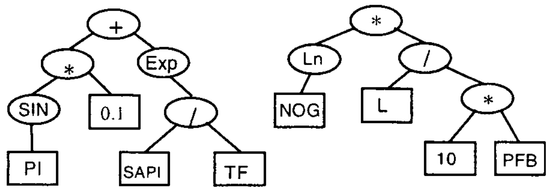

Fig.3.2 Two Parental Individuals

Assume that the points of both trees be numbered in a depth-first and left-to-right way. Suppose that the sixth point (out of the 9 points in Fig. 3.2 (a)) is randomly selected as the crossover point for the first parent. The crossover point of the first tree is therefore the Exp function. Suppose also that the fourth point (out of the 8 points in Fig. 3.2 (b)) is selected as the crossover point of the second parent. The crossover point of the second parent is therefore the * function. Note that because entire sub-trees are swapped and because of the closure property of the functions themselves, this genetic crossover (recombination) operation produces syntactically legal tree-type individuals as offspring in all situations.

After crossovers are performed, mutations take place. This is to prevent all solutions in population from falling into a local optimum of solved problem. Mutation changes randomly the new offspring. It is important in evolution because unlike crossover (which merely trades genes), new gene values are introduced. This further increases the diversity of the population members.

The conventional mutation operation for a binary string flips the bits in the string with a given probability. As to our solution, the typical mutation for a tree-type individual, on the other hand, would swap sub-trees with a given probability. In our solution, the mutation operator uses the already implemented crossover mechanisms.

(1) Randomly select a parental individual, which undergo mutation according to the population mutation probability.
(2) Create a temporary random derivation tree-type individual.
(3) Apply the crossover operator to the derivation tree to be mutated, using the temporary tree as second tree.

It is obviously seen in the figures that the mutation operator is implemented as crossover between the tree to be mutated on one side and a random tree on the other side.

In addition, some useful techniques such as "elitism" and " survival selection" have been used to make the fitness of the final results as higher as possible. The evolution process can be terminated whenever either a preset maximum number of generations G have been reached or the standardized fitness of some individual in the population equals zero or a preset small number Z.

IV. IMPLEMENTATION AND EXPERIMENTAL RESULTS

Our forecasting system consists of two parts: a subsystem that generates forecasting expression and a subsystem for real forecasting. User must first input some characteristic parameters for some specific circuits, These data are to be stored in the database, and then inputs testability parameter values of some test generation algorithm for these circuits. GA-based forecasting system calculates the forecasting expressions using genetic algorithms. User may set some parameter values such as initial population size, number of generations, crossover probability, crossover type, mutation probability, mutation type, number of delete individual, and termination condition, etc, according to their own need.

4.1 Implementation of Subsystems in Forecasting System

The forecasting system presented in this paper is programmed using Visual C++5.0. It is simple to operate and easy to use with a user-friendly interface. Besides, many mathematical function, i.e., exponent function, log function, and square root function, libraries can be used directly. MFC function libraries in Visual C++ 5.0 is helpful in writing right windows structure that provide convenience when user input data interactively and system outputs results.

ISCAS'85 and ISCAS'89 benchmark circuits are used for establishing the forecasting models for each test generation algorithm. Up to now, We have constructed the forecasting models for 18 algorithms. However, this system can be easily extended to forecasting other more algorithms.

4.2 Results

For the limit of space, results for only one of test generation algorithms for combinational circuits and one for the sequential circuit are listed here for reference.

4.2.1 *The FC forecasting model for the SOCRATES algorithm (for combinational circuits) is evolved into the following expression:*

$FC_{SOCRATES}$=(sqrt(cuberoot((power10(cbw(3.183)))/(ln(9.219)))))+((cos((((NPI)-(NIL))*(sin(FLT)))*((cos(3.570))/((INL)/(MFI)))))+(((((2.428)/(FOB))*((3.806)*(9.820)))*((((((exp(0.345))+(cbw(3.085)))*(sh((1.831)/(TNL))))-((sin((6.437)+(MFI)))-(((7.063)-(INL))+((INL)-(0.229)))))+(2.475))+((FOB)-(1.089))))-(quintroot(((2.728)-(FLT))*(1.570)))))

4.2.2 *TheTP forecasting model for the SOCRATES algorithm is evolved int o the following expression:*

$TP_{SOCRATES}$=((sin(quintroot((((MFO)+(AFI))*((((FOB)-(power10((MFI)*(3.426))))+((((NOG)/(INL))*((TNL)/(MFI)))/(cbw((2.572)*(FLT)))))*(cuberoot((((FOS)*(TNL))-((NPO)+(MFI)))/((cube(NOG))-((0.402)*(FOS)))))))/((cuberoot(FLT))+((AFI)*(NIL))))))*((((cbw(sin(AFI)))*((NPO)-(FOB)))-((quint(AFI))*(sin(4.435))))/(((6.626)+(1.212))+(((8.682)*(9.009))*((2.590)-(AFI))))))-((((th((6.047)*(INL)))+(cos((3.606)+(7.118))))-((((MFO)+(MFI))+(ch(AFI)))/(log10((2.808)*(AFI)))))*(quintroot((sqrt(quint(INL)))/(((FLT)+(NOG))*(MFO)))))

4.2.3 *The FC forecasting model for the HITEC algorithm (for sequential circuits) is evolved into the following expression:*

FC_{HITEC}=(exp(((th((((9.683)+(3.218))/(NSG))-(NOG)))/(((LFF)+(FLT))+((NSF)*((NPI)*(LFF)))))*(((9.865)/(cube((quintroot(NPI))-(cos(NOG)))))+(((4.154)*(NSG))-(((PFB)+(NOL))-(cube(SED))))))*((((cuberoot(((cos(FLT))*((NOL)*(FFL)))+(((SED)*(6.1

87))*(((sin(NPI))*(sqrt(FOS)))+(4.827)))))+((9.437)+(ISC)))+(((((7.107)+(
arctg((ISC)-(exp(SED)))))+(arctg((((NPO)-((cube((LFF)/(NFF)))+(((NFF)-
(7.625))-
(NSF))))*((quint(ISC))*(power10(quint(sin(NSG))))))/(FOS))))+(4.307))*(
(2.820)+(1.278))))-(((sin(NFF))+((FFL)-(3.924)))-
((sin(FOS))*(quintroot(NOL)))))

4.2.4 *The TP forecasting model for the HITEC algorithm is evolved into the following expression:*

TP_{HITEC}=((ln(sqrt((exp((exp((ln(NFF))/(sin(sqrt(exp(NPI))))))/(ln(SED))))*(
exp(cuberoot(((sqrt(sqrt(FLT)))*(((SED)*(NPI))*(ln(40))))*((cuberoot((arc
tg((ln(cuberoot((exp((exp((ln(NFF))-
(cuberoot(cuberoot(cuberoot(28))))))/(ln(SED))))*(exp(cuberoot(((sqrt(sqrt(
FLT)))*(((sqrt(sqrt(FOS)))*(23))*(ln(((11)*(exp((sin(sin((FLT)*(sqrt((NFF)
-(arctg(17)))))))/(sin(SED)))))*(sin(FFL))))))*((sin((arctg((33)-
(NOL)))+(cuberoot((exp(((FFL)+(NFF))*((NOF)-
(cuberoot(sqrt((34)*(FOS)))))))/(((67)+((sin(NFF))-
(4)))*((NOG)*(exp(ln(36)))))))))-
(sin(cuberoot(sin(33)))))))))))*(1)))+(arctg((exp((cuberoot((NOG)/(NFF)))*
((NOF)-(ln((FLT)*(NSF))))))*((sin(LFF))/(FLT))))))-
(sin(1)))))))))*((cuberoot(((cuberoot(ln((exp((33)*(sin(NOL))))+(exp(sqrt(l
n((sin((59)*(ln(ISC))))/(arctg(exp(cuberoot(((ln(exp(((cuberoot(1))+(sqrt(s
qrt(arctg(SED)))))/(ln(ln(arctg(FLT)))))))*((cuberoot(((sqrt(cuberoot(FLT)))
+((((sqrt(ln((FLT)*(FOS))))+(cuberoot((NOG)/(exp(20)))))*(ISC))+(sqrt(F
OS))))-
((sqrt(sqrt((FLT)*(NOG))))/(sin((arctg((sin((exp(FOS))*(exp(sqrt(sin(arctg
((NSG)*(sqrt((ISC)*(NOG))))))))))-
(sin(sqrt(NPI)))))+(ln(ISC)))))))*(exp(sqrt(exp(sqrt((NOG)/(57))))))))+(sin(
(exp(PFB))-((NOF)-
(FLT)))))))))))))))*(((cuberoot(((cuberoot(sqrt(NPO)))*(((SED)*(NPI))*(ar
ctg(NFF))))/((sin(arctg((FOS)/(8))))+(sin((sin((ln(arctg(1)))*(cuberoot(ln(N
OL)))))+(exp(NPI))))))))*(NPI))-(LFF)))-
(arctg((((NOG)*(NOF))+(ISC))+((FOS)-
(NSG))))))*(exp(cuberoot(sin(cuberoot((FLT)/(62))))))))*(arctg(sqrt((exp((
((exp(NPI))/(exp(PFB)))/(cuberoot(sqrt(SED))))/(ln((exp(arctg((LFF)-
(exp((FOS)*(ln(exp(sin((exp((ln(arctg(NSG)))*(ln(ln(39)))))+(exp((NSG)/(
FLT))))))))))))*((sqrt(exp(4)))-
(cuberoot(exp(arctg(NSF))))))))))/(cuberoot(sqrt(ln(ISC))))))

Both actual and forecasted values of fault coverage (FC) and the number of test patterns (TP) for the ISCAS'85/ISCAS'89 benchmark circuits by using SOCRATE and HITEC algorithms have been omitted in this paper for space limitation.

V. CONCLUSIONS

We have constructed a testability parameter forecasting system for some well-known test generation algorithms to forecast their test efficiency based on our genetic algorithm model presented in this paper. Testability forecasting for existing algorithms is so useful and helpful for those who intend to test a large amount of circuits (especially for VLSI/ULSI circuits) without knowing the knowledge of which test generation algorithm should be correctly employed, i.e. the best one, so that a significant testing cost can be reduced. The key point of this system is to establish an adequate as well as an accurate model for each algorithm so that it may make a precise prediction for its test efficiency. A variety of models have been tried for algorithms concerned, but so far the genetic algorithm model could be considered as the best one for its non-linearity.

Theoretically, we may establish a genetic algorithm model for any test generation algorithm. However, it takes some time for people to have a good model for all the existing algorithms being widely used. Therefore, the process of establishing an adequate model for as many algorithms as possible still leaves to be desired.

REFERENCES

[1] Johnson, B. W., *Design and Analysis of Fault-Tolerant Digital Systems*, Reading, Mass.: Addison-Wesley Pub. Co., 1989.

[2] Roth, J.P., *Diagnosis of Automata Failures: A Calculus and a Method*, IBM Journal of Research and Development, Vol.10, No.4, pp.278-291, July 1966.

[3] Roth, J.P., Bouricius, W.G., and Schneider, P.R., *Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits*, IEEE Trans. on Electronic Computers, Vol.EC-16, No.10, pp.567-579, October 1967.

[4] Abramovici, M., Breuer, M.A., and Friedman, A.D., *Digital Systems Testing and Testable Design*, Computer Science Press, 1990.

[5] Schneider, R.R., *On the Necessity to Examine D-Chains in Diagnostic Test Generation*, IBM Journal of Research and Development, Vol.11, No.1, pp.14, January 1967.

[6] Goel, P., *An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits*, IEEE Trans. on Computers, Vol.C-30, No.3, pp.215-222, March 1981.

[7] Fujiwara, H., and Shimono, T., *On the Acceleration of Test Generation Algorithms*, IEEE Trans. on Computers, Vol.C-32, No.12, pp.1137-1144, December 1983.

[8] Cha, C.W., Donath, W.E., and Ozguner, F., *9-V Algorithm for Test Pattern Generation of Combinational Digital Circuits*, IEEE Trans. on Computers, Vol.C-27, No.3, pp.193-200, March 1978.

[9] Muth, P., *A Nine-Valued Circuit Model for Test Generation*, IEEE Trans. on Computers, Vol.C-25, No.6, pp.630-636, June 1976.

[10] Kirkland, T., and Mercer, M.R., A Topological Search Algorithm for ATPG, Proc. 24th Design Automation Conf. pp.502-508, June 1987.

[11] Abramovici, M., Kulikowski, J.J., Menon, P.R., and Miller, D.T., SMART and FAST: Test Generation for VLSI Scan-Design Circuits, IEEE Design & Test of Computers, Vol.3, No.4, pp.43-54, August 1986.

[12] Shiyi Xu and T.J Frank, *An Evaluation of Test Generation Algorithms for Combinational Circuits*, Proc. IEEE The Eighth Asian Test Symposium (ATS'99), pp.63-69, Nov. 1999.

[13] Shiyi Xu and Dias, G. P., *Testability Forecasting for Sequential Circuit*, Proc. IEEE The fourth Asian Test Symposium (ATS'95), pp.199-205, Nov. 1995

[14] Hu Yang, *Testability Forecasting for Combinational Circuits Using Neural Networks*, Masters Degree Thesis, Shanghai University, 1997.

[15] Shiyi Xu, Peter Waiganjo, and Dias, G.P., *Testability Prediction for sequential Circuits Using Neural Networks*, Proc. IEEE The Sixth Asian Test Symposium (ATS'97), pp.126-132, Nov. 1997

[16] Mahfoud, S. and Mani, G., *Financial Forecasting Using Genetic Algorithms*, Applied Artificial Intelligence, Vol. 10, Num. 6, pp. 543-565, 1996.

[17] Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Berlin: Springer-Verlag, 1994.

[18] Russell, S. J., and Norvig, P., *Artificial Intelligence: a modern approach*, Englewood Cliffs, N.J.: Prentice Hall, Inc., pp.619-620, 1995.

[19] Welstead, S. T., *Neural Network and Fuzzy Logic Applications in C/C++*, New York: John Wiley & Sons, Inc., pp. 283-304, 1994.

[20] Schwefel, H.-P., *Numerical Optimization for Computer Models*, John Wiley, Chichester, UK, 1981.

[21] Fogel, L.J., Owens, A.J., and Walsh, M.J., *Artificial Intelligence Through Simulated Evolution*, John Wiley, Chichester, UK, 1966.