GA Based Test Case Generation Approach for Formation of Efficient Set of Dynamic Slices

Debasis Mohapatra
Dept. of CSE
PMEC(A constituent college of B.P.U.T)
Berhampur,India
devdisha@gmail.com

***Abstract*:-Automated test case generation is an efficient approach for software testing. Slicing of program provides ease to testability and enhances debugging capacity. To generate the dynamic slice, slicing criterion is required in which the input data parameter is the essential component. Most of the research work focuses on deriving the input by random consideration but it simply takes a longest period of time to generate slices that provides the path coverage of Unit Under Test (UUT). This paper generates the optimal test cases by using Genetic Algorithm (GA) and Control Flow Graph (CFG), these test cases cover all the independent path present in the CFG. The optimal test cases are supplied as input component of the dynamic slicing criteria. So the dynamic slice criteria that use these optimal test cases as the input generates the efficient dynamic slice set that is helpful in efficient testing and efficient debugging. Here two approaches, first the dynamic slice using node marking and the second by using relevant sets are discussed according to optimal test cases as input component.**

Keywords: Automated test case generation, Dynamic slicing, Genetic Algorithm, Test suite, Sampling

I. INTRODUCTION

Automation is the most emerging trend in software engineering that reduces the effort in a huge factor. Testing is an important part in software development. Test case generation is a prime step to achieve testing of the Software Under Test (SUT). Automation in test case generation reduces the effort involved in testing. A good test suite uncovers all most all bugs present in the software. Automated testing is a key factor to attain a high level of reliability. Evolutionary approach like GA is an optimization technique used to generate optimal test cases of the UUT[7,22,24]. This paper focuses on the generation of optimal test cases by using Genetic Algorithm for a control flow graph representation of UUT. As the cyclomatic complexity represents the possible independent path of the CFG [1], it is used here to find out the probability of getting successful test suite. The dynamic slices are usually evaluated at the end point of the program [14,15,16] but the input parameter is always considered as a random choice[9,10,19]. Rather considering the input parameter as random, the optimal test cases are taken as input parameter that reduces the time complexity of finding out effective set of slices that covers all paths according to different variables of program. This efficient dynamic slice set provides a high degree of testability and debugging. Program slicing is used in various applications like refactoring, parallelization of program, regression testing, maintenance, program verification [19]. Node marking method is used in this paper to find out the dynamic slices that use Program Dependence Graph (PDG) and relevant sets method is used that takes the help of execution history to find out dynamic slices.

II. TEST HARNESS

Test harness Patterns are used to provide automation in generating test cases[20]. Binder explains different test harness pattern in which test cases, test control, test drivers, test framework are the important components[20]. The pattern that is adopted in this paper uses Test case/suite method.

III STEPS OF GENETIC ALGORITHM

Genetic Algorithm is an evolutionary optimization Technique. Now a day there is a huge applications of evolutionary approaches is noticed in the field of Software Engineering. Most of the testing problems can be solved by simulating the problem as a search based problem. And the Evolutionary approaches like Genetic Algorithm is an efficient method for solving this type of problem. Path testing is a white box testing technique that is converted to

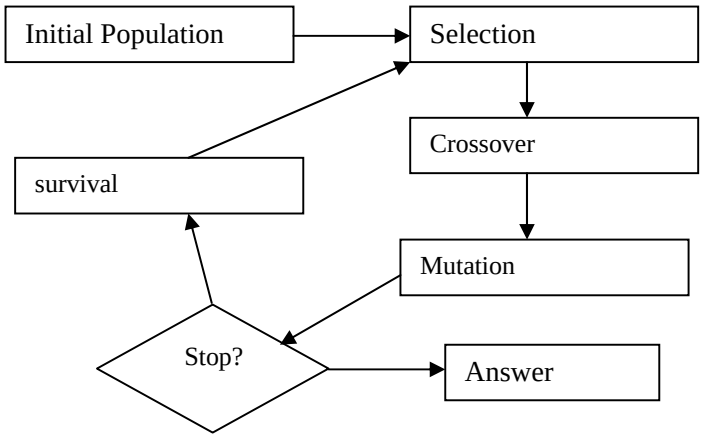

Fig.1 GA steps

the search based problem and solved by GA. Fig.1 depicts all the steps of GA and the links show the order of execution of steps.

A) Initialization: This genetic operator creates an initial population of chromosomes, at the beginning of the genetic algorithm execution. Usually initialization is random.

B) Selection: The selection operator is used to choose chromosomes from a population for mating. This mechanism defines how these chromosomes will be selected, and how many offspring each will create. The expectation is that, like in the natural process, chromosomes with higher fitness will produce better offspring. Selection has to be balanced: too strong selection means that best chromosomes will take over the population reducing its diversity needed for exploration; too weak selection will result in a slow evolution. Classic selection methods are Roulette-wheel, Rank based, Tournament, Uniform, and Elitism.

C) Crossover: The crossover operator is practically a method for sharing information between two chromosomes; it defines the procedure for generating an offspring from two parents. The crossover operator is considered the most important feature in Genetic Algorithm; especially where building blocks exchange is necessary. One of the most common crossover operators is Single-point crossover in which position is chosen at random and the elements of the two parents before and after the crossover position are exchanged.

D) Mutation: The mutation operator alters one or more values of the allele in the chromosome in order to increase the structural variability. This operator is the major instrument of any particular area of the entire search space.

E) Survival: Survival step is required to choose the chromosomes for next generation. It is not always mandatory to work out this phases. This phase is needed for selecting the chromosomes from parent population as well as children population by fitting some random numbers.

IV. APPLICATION OF GA IN SOFTWARE TESTING

Zhao and Shanshan have developed a model by using neural networks that simulates the functionality of SUT. They have devised an efficient algorithm to find corresponding inputs of particular output[28]. Gupta and Rohil have proposed a new approach for test case generation of object oriented software using GA[29]. Last, Eyal and kandel have proposed a new computationally intelligent approach to generation of effective test cases based on a novel, Fuzzy-Based Age Extension of Genetic Algorithms (FAexGA)[30]. Phil McMinn, Mark Harman, David Binkley and Paolo Tonella have employed a verification and validation technique by the help of GA[31].

V. BASIS PATH TESTING

Tom Mc-Cabe first proposed basis path testing approach. The basis path method is a white box testing technique and helps the test case designers to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths. Test cases derive to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing.

A) Control Flow graph:- Control Flow Graph is the graphical notation G(N,E) in which N is the set of nodes that represent the statements and E is the set of edges that represents transfer of control between nodes.

*B) Cyclomatic Complexity***:** The logical complexity of the program can be estimated by using cyclomatic complexity[1]. In case of basis path testing, cyclomatic complexity defines the number of independent path in the basis set of a program and provides an upper bound for the number of test that must be conducted to ensure that all statements is to be executed at least once. An independent path is any path through the program that includes at least a new processing statement or a new condition. When stated in terms of a flow graph, an independent path must move along at least one edge that has not been traversed before the path is defined. Cyclomatic complexity has a foundation in graph theory and provides us with extremely useful software metric. Cyclomatic complexity is computed by using the following formulae [26].

Cyclomatic complexity for a graph G is defined as:

$CC(G) = E - N + 2$ (i)

Where 'E' is the number of graph edges and 'N' is the number of graph nodes.

$CC(G) = P + 1$ (ii)

Where 'P' is the number of predicate nodes contained in the graph G.

VI. MEASURING PROBABILITY OF GETTING SUCCESSFUL TEST SUITE

A successful test suite in path testing is the test suite that covers all the independent paths of the CFG else called unsuccessful test suite.

A) Stratified Sampling: In Stratified Sampling all the data are grouped according to there satisfaction of group criterion[27]. Here the data are equivalent to test cases and the group criteria are equivalent to conditions of the UUT. Let us consider ' n' independent paths are present in CFG. Each with M_1, M2… Mn numbers of test cases.

Independent path<1>	Independent Path<2>	……	Independent Path<n>
M_1(test cases)	M_2(test cases)		M_n(test cases)

Total number of successful test suites=$M_1 * M_2 * \ldots\ldots\ldots * M_n$ (iii)

B) Random Sampling: In case of random sampling the sample units (test cases) are selected at random and the drawback of purposive sampling is completely overcome[27]. A random sample is one in which each unit of population has an equal chance of being included in it. Suppose we take a sample of size' n' from a finite population of size N. Then there are C(N,n) possible samples. A sampling technique in which each of the C(N,n) samples has an equal chance of being selected is known as random sampling and the sample obtained by this technique is termed as a random sample. We have used random sampling for extracting different test set from the optimized test cases generated by genetic algorithm, having the sample size equal to the cyclomatic complexity.

Number of test suite with T number of test cases is Q=C(N,T) (iv)

Where N=number of optimized test cases

T= Cyclomatic complexity

Probability of getting successful test suite=$(M_1*M_2*\ldots\ldots\ldots*M_n)/Q$ (v)

Where $Q=C((M_1+M_2+\ldots\ldots+M_n),T)$

VII. DYNAMIC SLICING

Program slice (p') is a part of the program(p) that executes independently. Program slice is projected from the program according to the slicing criterion. Dynamic slice of an UUT is based on the slicing criterion<S, V, I>. Here 'S' represents statement number, 'V' represents variable name and 'I' represents input test case. The dynamic slice is evaluated for a particular execution so it is a subset of static slice with same S and V[9,15]. Program Dependence Graph (PDG) and Dynamic Dependence Graph (DDG) are two imperative tools used in dynamic slicing [17]. Node making, Edge marking are the methods that are used in PDG or DDG [15]. Relevant sets method derives static slice but here it is used in deriving dynamic slice. The terminologies and their descriptions are defined that are used in this paper.

A) *Program Dependence Graph (PDG):* PDG is a directed graph(V,E) where 'V' represents vertices of a graph that in turn a statement of the program and 'E' represents control or data dependency between the nodes that is a graphical representation of control and data dependency between the statements.

B) *Node Marking method:* In node marking method the projection of the PDG with respect to nodes is taken according to the execution history then set of the statements are taken that are reachable from the considered node for slicing. It obeys a backward slicing approach.

C) *Relevant sets method for dynamic slicing:* In this method first the projection of the program is taken according to the execution history then RF(n),DF(n),Control(n)and Relevant(n) are measured. Relevant (n) varies according to the variables.

RF(n)-The set of variables that are used/referenced in statement 'n'.

DF(n)- The set of variables that are defined in statement 'n'.

Relevant(n)- if 'x' is the predecessor statement of 'y' then Relevant(x)={Relevant(y)-DF(x)}U{RF(x) if Relevant(y)∩DF(x)≠ ǿ} .

It is also a backward slicing method. It starts form the last statement and include all those statements that defines it relevant set variable, the process continues until the first statement reach. All included statements combine to construct the dynamic slice with respect to criterion<S, V, I>.

VIII. PROPOSED STRUCTURE

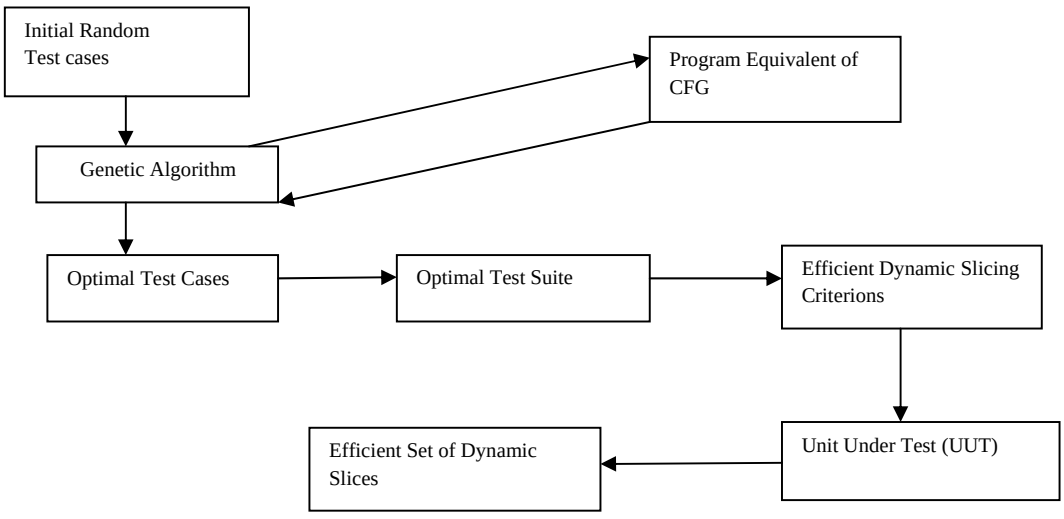

Fig.2-Proposed Structure

The structure [Fig. 2] depicts the work flow of the approach adopted to generate an efficient set of dynamic slices. Initial process starts with randomness by supplying random test cases to Genetic Algorithm. The GA compares the test cases with the path conditions of the CFG and going on modifying the test cases until getting the test cases that discover all the paths of the CFG called as optimal test cases. The optimal test suites are created from the optimal test cases by applying stratified sampling statistics. Efficient Dynamic Slicing Criteria are formulated from one of the optimal test suite. These set of criteria are implemented on the UUT to yield an efficient set of dynamic slices.

IX. PROPOSED ALGORITHM FOR OPTIMAL TEST SUITE GENERATION

```
Optigenetic(n, Fm, popsize , Imax ,Pc,Pm, t,)
{Input:
the function mlodel Fm
Number of input variables n
population size popsize
Maximum iteration number Imax
Crossover probability Pc
Mutation probability Pm
Number of leaf nodes t
Output:
optimal test suites
I=0;
Popx[i]←TestHarness(popsize);
c=0;
while(i!=Imax|| kc == t)
{
Choosesrv();
Boolean X= graphequi(srv);
if(X==true)
{
c++;
Changesrv();
}
else
{
Fitness[i]   fitnesseval(popx[i]);
Population   select(popx[i]; fitness[i]);
Popx[i]   crosssover(population; fitness; Pc);
Popx[i]   mutate(population; fitness; Pm);
}i++;
}Optimal( popx[i]);
}
```

CFG is represented by an equivalent program (*graphequi(srv)*) and the search node as input that is the one of the leaf node. GA optimizes the test cases to satisfy the search nodes one by one. It is applied on the program structure where the paths obey the property of trichotomy. Some of the test cases are passed to the test harness that in turn passes all the test cases to the genetic algorithm as initial population. The genetic algorithm then optimizes the test cases by using selection, crossover and mutation operators. The optimized test cases are then passed through stratified sampling (*optimal(popx[i])*) to derive the optimal test suites.

X. IMPLEMENTATION DETAIL AND RESULT

A) Schematic Representation of UUT: Schematic representation of the program a program where all the actual functions are not called rather a schema is called like f1(a,b),g1(a,b) etc.

```
1.  Read(a,b);
2.  if(a<b)
3.     y=f1(a,b);
4.     x=g1(a,b);
5.  else if(a==b)
6.     y=f2(a,b);
7.    x=g2(a,b);
8.  else
9.     y=f3(a,b);
10.    x=g3(a,b);
11. write(x);
12. write(y);
```

B) Contol flow graph of UUT

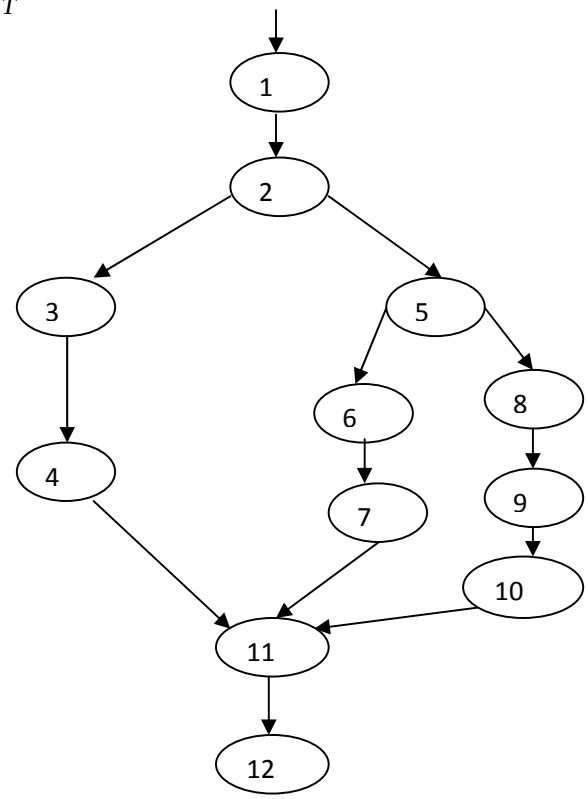

Fig.3 CFG of UUT

C) Program equivalent of CFG:

```
if(a<b)
   printf("1,2,3,4,11,12");
else if(a==b)
   printf("1,2,5,6,7,11,12");
else
    printf("1,2,5,8,9,10,11,12");
```

D) Selecting Operators of GA:

Fitness evaluation (F): F(a,b)=1/(|a-b|+.04)^3 Where *a, b* are the parameters of test condition. Constant .04 is added to provide a value for equal condition.

Selection procedure:- One of the parents is selected from the chromosomes having highest fitness value and another from lower fitness values.

Crossover operator:- Here, we have used the single crossover operator. Crossover probability (Pc) is considered as 0.4.

Mutation:- In a binary code, mutation simply means changing the state of gene from 0 to 1 or vice versa. Here mutation probability (Pm) is considered as 0.3.

E) Optimal Test cases

Population Table(Evaluated in %15)

INITIAL POPULATION					FINAL POPULATION				
a	b	Chromosome	Fitness	Path/cond	a	b	Chromosome	Fitness	Path/cond
12	4	11000100	0.0155	1,2,5,8,9,10,11,12 /a>b	1	1	00010001	2500	1,2,5,6,7,11,12 /a==b
111	45	01100000	0.0276	1,2,5,8,9,10,11,12 /a>b	111	45	01100000	0.0276	1,2,5,8,9,10,11,12 /a>b
120	3	00000011	0.1096	1,2,3,4,11,12 /a<b	11	2	10110010	0.0123	1,2,5,8,9,10,11,12 /a>b
11	7	10110111	0.0619	1,2,5,8,9,10,11,12 /a>b	2	3	00100011	0.9612	1,2,3,4,11,12 /a<b

F) Optimal Test Suites

Stratified sampling Table

Independent path1 1,2,5,8,9,10,11,12/a>b	Independent path2 1,2,3,4,11,12/a<b	Independent Path3 1,2,5,6,7,11,12/a==b
<111,45> <11,2>	<2,3>	<1,1>

Number of Successful test suites	Probability of getting successful test suite	Optimal Test Suites
2*1*1=2	2/C((2+1+1),3)=2/4=.5	Test Suite1-<1,1><111,45><2,3> Test Suite2-<1,1><11,2><2,3>

G) Efficient Dynamic slice criteria

Efficient Dynamic Slice criterion can be constructed by taking any one test suite .Let us consider the test suite as<1,1><111,45><2,3>

<x,11,a=1,b=1>	<y,12,a=1,b=1>
<x,11,a=111,b=45>	<y,12,a=111,b=45>
<x,11,a=2,b=3>	<y,12,a=2,b=3>

H) Efficient dynamic slice formation by using Node marking in PDG:

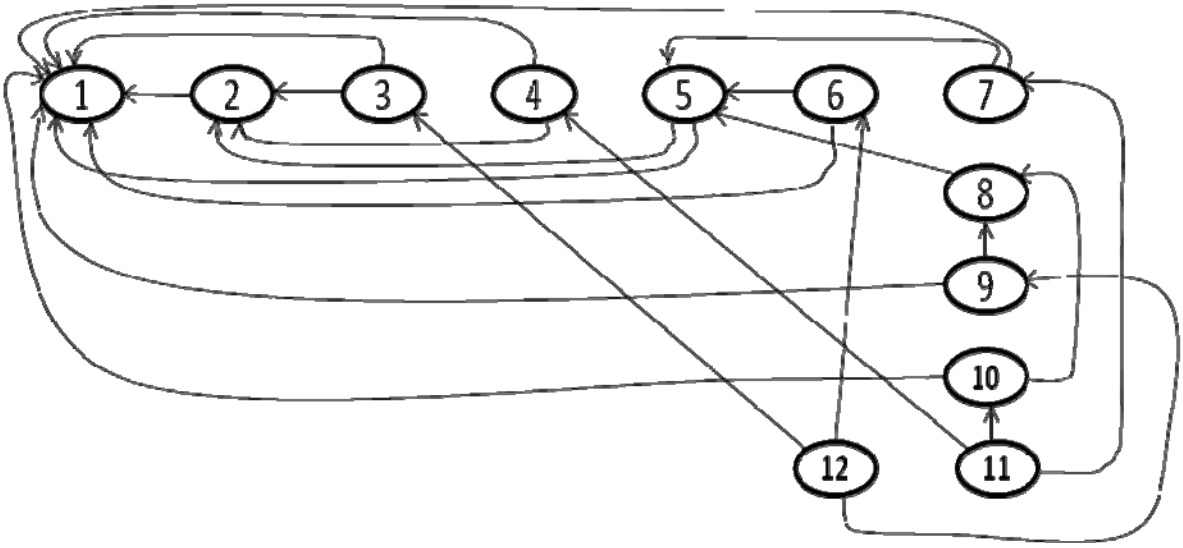

Fig.4 PDG of UUT

Adjacency Matrix Representation of PDG

From/To	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0	0	0
4	1	1	0	0	0	0	0	0	0	0	0	0
5	1	1	0	0	0	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0	0	0
7	1	0	0	0	1	0	0	0	0	0	0	0
8	0	0	0	0	1	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	1	0	0	0	0
10	1	0	0	0	0	0	0	1	0	0	0	0
11	0	0	0	1	0	0	1	0	0	1	0	0
12	0	0	1	0	0	1	0	0	1	0	0	0

Slice Criterion	Execution History	Dynamic Slices
<x,11,a=1,b=1>	<1,2,5,6,7,11,12>	<1,2,5,7,11>
<y,12,a=1,b=1>	<1,2,5,6,7,11,12>	<1,2,5,6,12>
<x,11,a=111,b=45>	<1,2,5,8,9,10,11,12>	<1,2,5,8,10,11>
<y,12,a=111,b=45>	<1,2,5,8,9,10,11,12>	<1,2,5,8,9,12>
<x,11,a=2,b=3>	<1,2,3,4,11,12>	<1,2,4,11>
<y,12,a=2,b=3>	<1,2,3,4,11,12>	<1,2,3,12>

I) Efficient dynamic slice formation by using relevant sets method

Projection from program according to execution history<a=1,b=1>	RF(n)	DF(n)	Control(n)	Relevant(n) w.r.t(x)	Relevant(n) w.r.t(y)
1.Read(a,b)		a,b			
2. if(a<b)	a,b			a,b	a,b
5. else if(a==b)	a,b		2	a,b	a,b
6.y=f2(a,b)	a,b	y	5	a,b	a,b
7.x=g2(a,b)	a,b	x	5	a,b	y
11.write(x)	x			x	y
12.write(y)	y				y
Dynamic Slice w.r.t. x=<1,2,5,7,11>			Dynamic Slice w.r.t y=<1,2,5,6,12>		

Projection from program according to execution history<a=111,b=45>	RF(n)	DF(n)	Control(n)	Relevant(n) w.r.t(x)	Relevant(n) w.r.t(y)
1.Read(a,b)		a,b			
2. if(a<b)	a,b			a,b	a,b
5. else if(a==b)	a,b		2	a,b	a,b
8.else			5	a,b	a,b
9.y=f3(a,b)	a,b	y	8	a,b	y
10.x=f3(a,b)	a,b	x	8	a,b	y
11.write(x)	x			x	y
12.write(y)	y				
Dynamic Slice w.r.t. x=<1,2,5,8,10,11>			Dynamic Slice w.r.t y=<1,2,5,8,9,12>		

Projection from program according to execution history<a=2,b=3>	RF(n)	DF(n)	Control(n)	Relevant(n) w.r.t(x)	Relevant(n) w.r.t(y)
1.Read(a,b)		a,b			
2. if(a<b)	a,b			a,b	a,b
3.y=f1(a,b)	a,b	y	2	a,b	a,b
4.x=g1(a,b)	a,b	x	2	a,b	y
11.write(x)	x			x	y
12.write(y)	y				y
Dynamic Slice w.r.t. x=<1,2,4,11>			Dynamic Slice w.r.t y=<1,2,3,12>		

XI. CONCLUSION & FUTURE WORK

This paper describes search based approach for test case generation .Genetic Algorithm decreases the probability of being trapped in local optimal values because it uses various chromosomes that works simultaneously. The dynamic slices of the program represent different projections of the program. In efficient set of dynamic slices no two slice signify the same set of statement coverage. So it is an effective method for automated testing. Sequential version of GA is quite slower in generating the test cases of complex software. This work can be extended to parallel domain

by using Parallel Genetic Algorithm and Parallel slicing approach that will reduce the time bound for generating efficient set of dynamic slices for complex software.

REFERENCES

[1] Roger S. Pressman: "Software Engineering", A Practitioner's Approach 5th Edition, Mcgraw Hill, 1997.
[2] DVLN Somayajulu,Ajay Kumar Bothra, Prashant Kumar, Pratyush ,"An Efficient Slicing Approach For Test Case Generation ".
[3] Mitrabinda Ray, Soubhagya Sankar Barpanda, Durga Prasad Mohapatra,"Test Case Design Using Conditioned Slicing Of Activity Diagram",International Journal Of Resent Trends In Engineering,Vol.1,No.2,May 2009.
[4] Andreas Leitner,Manuel Oriol,Andreas Zeller,Ilinca Ciupa,Bertrand Meyer,"Efficient Unit Test Case Minimization", ASE'07, November 4–9, 2007, Atlanta, Georgia, USA.
[5] P. Maragathavalli, M. Anusha, P. Geethamalini, S. Priyadharsini,"Automatic Test-Data Generation For Modified Condition/ Decision Coverage Using Genetic Algorithm",International Journal Of Engineering Science And Technology, Vol. 3 No. 2 Feb 2011, Pp-1311-1318.
[6] Premal B. Nirpal And K. V. Kale,"Comparison Of Software Test Data For Automatic Path Coverage Using Genetic Algorithm", Internationa Journal Of Computer Science & Engineering Technology (IJCSET), Vol. 1 No. 1,Pp 12-16.
[7] Raquel Blanco, Javier Tuya, Belarmino Adenso-Díaz,"Automated Test Data Generation Using A Scatter Search Approach",Information And Software Technology,April 2009.
[8] Bruno T. De Abreu1_, Eliane Martins1 , Fabiano L. De Sousa," Automatic Test Data Generation For Path Testing Using A New Stochastic Algorithm".
[9] Durga Prasad Mohapatra, Rajib Mall And Rajeev Kumar," An Overview Of Slicing Techniques For Object-Oriented Programs",Informatica(2006),Pp253-277.
[10] Durga Prasad Mohapatra,Madhusmita Sahu,Rajeev Kumar , Rajib Mall," Dynamic Slicing Of Aspect-Oriented Programs",Informatica(2008),Pp 261-274.
[11] Frank Tip,"A Survey Of Program Slicing Technique",The Merriam-Webster Dictionary.
[12] Jens Krinke," Advanced Slicing Of Sequential And Concurrent Programs",April 2003.
[13] "Dependence Graphs And Program Slicing", *Codesurfer Technology Overview.*
[14] David W.Binkley,Keith Brain Callagher,"Program Slicing".
[15] Keith Gallaghe David Binkley,"Program Slicing".
[16] Aharon Abadi Ran Ettinger Yishai A. Feldman,"Fine Slicing For Advanced Method Extraction",IBM Haifa Research Lab,2009/8/27.
[17] Andrea De Lucia,"Program Slicing: Methods And Applications".
[18] Xiangyu Zhang Rajiv Gupta," Cost Effective Dynamic Program Slicing", *PLDI'04,* June 9–11, 2004, Washington, DC, USA.
[19] Hiralal Agrawal,Joseph R.Horgan,"Dynamic Slicing",ACM SIGPLAN Notice,Vol.25,No.6,Pp-246-256,June 1990.
[20] Binder R.V,"*Testing Object-Oriented System Models, Patterns, And Tools*", Pp. 957-1017.
[21] Ruilian Zhoa, Shanshan Lv,"*Neural-Network Based Test Cases Generation Using Genetic Algorithm*", 13th IEEE International Symposium On Pacific Rim Dependable Computing.2007.
[22] Nirmal Kumar Gupta And Dr. Mukesh Kumarrohil,"*Using Genetic Algorithm For Unit Testing Of Object Oriented Software*", First International Conference On Emerging Trends In Engineering And Technology 2008.
[23] Bruno T.De Abreu, Eliane Martins, Fabiano L.De Sousa "*Automatic Test Data Generation For Path Testing Using A New Stochastic Algorithm*". 2004.
[24] Mark Last1, Shasy Eyal1, And Abraham Kandel"*Effective Black-Box Testing With Genetic Algorithms*", 2000.
[25] Paul C. Jorgensen," *A Craftsman's Approach*", Second Edition, Pp.448-515.2002.
[26] S. C. Gupta, V. K. Kapoor," *Fundamental Of Mathematical Statistics"*, Eleventh Edition.,Pp.5.1-5.72, 2002.
[27] Ruilian Zhoa,Shanshan Lv, "Neural-Network Based Test cases Generation using Genetic algorithm",13th IEEE International Symposium on Pacific Rim Dependable Computing.2007.
[28] Gupta N. K and Rohil M.K, "Using genetic Algorithm For Unit Testing of Object Oriented Software",First International conference on Emerging Trends in Engineering and Technology,2008.
[29] Mark Last, Shasy Eyal, and Abraham Kandel, "EffectiveBlack-Box Testing with Genetic Algorithms".
[30] Phil McMinn ,Mark Harman, David Binkley and Paolo Tonella, "The Species per Path Approach to SearchBased Test Data Generation", ISSTA '06, July 17-20,Portland, Maine, USA.

AUTHORS PROFILE

Debasis Mohapatra is a lecturer in the Dept. Of CSE, in Parala Maharaja Engineering College*(A constituent college of B.P.U.T).* His research interest is "application of evolutionary approach in Software Engineering domain".