Basis Path Testing using Genetic Algorithm: A Survey

Deepak Garg\textsuperscript{1} Dr. Sanjay Tyagi\textsuperscript{2}
\textsuperscript{1} M.Tech Scholar,\textsuperscript{2} Assistant Professor
\textsuperscript{1,2} Department of Computer Science and Applications
\textsuperscript{1,2} Kurukshetra University Kurukshetra, India

Abstract—Software testing is the most significant analytic quality assurance for software products, but it is very expensive and time consuming process. This limitation is overcome by automatic testing to reduce high cost, to increase reliability and efficiency as compared to manual testing. Basis path testing is a coverage criterion of software testing that can detect almost 65 percent of errors in program under test. In this paper a survey of basis path testing using genetic algorithm (GA) has been presented. Two most important fitness functions (branch distance and approximation level) have also been presented to transform test data generation problem into optimization problem. On the basis of this study, new approaches or improvements in GA can be used to generate test data automatically.

Key words: Approximation Level, Basis Path Testing, Branch Distance, Genetic Algorithm, Software Testing

I. INTRODUCTION

A. Background:

In general, manual software testing accounts for approximately 50 percent of the elapsed time and more than 50 percent of the total cost in software development [1, 2] & automated software testing is a promising way to cut down time and cost.

Path oriented test data generation is an undecidable problem [3]. On the whole, test data generation methods can be classified into two types: Static Methods and Dynamic Methods. Static methods include domain reduction [4, 5] and symbolic execution [6] etc. Static methods may get into trouble when they handle indefinite arrays, loops, pointer references and procedure calls [7]. Dynamic methods include random testing, local search approach [8], goal oriented approach [9], chaining approach [10] and evolutionary approach [11, 12, 13]. As values of input variables are determined when programs execute, dynamic test data generation can avoid those problems that are confronted with the static methods.

As a robust search method in complex spaces (robust to dynamic), genetic algorithm was applied to test data generation in 1992 [11] and this evolutionary approach has been in interest since then. Branch distance based fitness function and approximation level based fitness function [14] were both used for GA based path oriented test data generation. Comparison of these two fitness functions for GA based path oriented test data generation [15] was also done. To improve the performance by parallelism, multi population genetic algorithm [16] was introduced. Antariksha Bhaduri [17] proposed a hybrid genetic algorithm based on genetic algorithm and artificial immune network algorithm (GAIN) for finding optimal collision free path in case of mobile robot moving in static environment filled with obstacles. Malin Bjomdotter [18] proposed a memetic algorithm for feature selection in volumetric data containing spatially distributed clusters of informative features in neuroscience application.

B. Need of Basis Path Testing and GA:

Basis path testing strategy can detect almost 65 percent of errors in program under test [19] and various structural test data generation problem can be transformed into a path oriented test data generation problem [20]. Furthermore, a path coverage criterion covers statement and branch coverage. So, basis path testing is selected in this paper as coverage criteria for software testing.

Many automatic tools for test data generation are already present (for generation of test data), but they are not good for large scale problems as they require knowledge of solution space and are also not robust to dynamic. Furthermore related works [7, 21] indicate that GA-based test data generation outperforms other dynamic approaches and static approaches. So, Genetic Algorithm is used in this paper for generating test data set from a pool of randomly generated test data automatically [22 , 23].

C. Objectives:

Since the basis path testing is more rigorous and practically more effective at detection of errors than other common unit test criteria. That’s why in this paper, it is presented that how test data is generated for basis path testing criterion by using genetic algorithm.

II. BASIS PATH TESTING

Basis path testing was first proposed by Tom Mc Cabe [20]. It is a white box testing technique. The basis path method enables the test case designer 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 [20]. This set of independent paths is called basis set. That’s why it is termed as basis path testing.

A. Control Flow Graph:

The control flow graph presents a skeletal model of all execution paths of a program [24]. In a flow graph, the nodes represent computational statements or expressions and edges represent transfer of control between nodes. Each possible execution path from the entry point to the exit point in a program unit has a corresponding path from the source node to the sink node in its flow graph.

B. Basis Set:

Basis set is a finite set of line arly independent paths through a standard flow graph [20]. An independent path is a path of a program, where at least one edge of this path never appears in any other path in the control-flow graph (CFG). Every path in the Basis set should satisfy these three conditions:

– Every path should be an independent path.
- All edges in a CFG should be covered by all paths in the basis set.
- Every path not contained in this basis set of paths can be constructed by linear operations among paths in this set.

Cyclomatic complexity number gives the number of independent paths (as basis set) as upper bounds for the number of tests that must be conducted to ensure that all the statements and every condition has been executed on its true and false sides at least once [24]. Table I gives cyclomatic complexity of any graph g, where V(g) is cyclomatic number, e is number of edges, n is number of nodes, d is number of decisions, r is number of regions & p is number of components of whole graph.

<table>
<thead>
<tr>
<th>Not Strongly connected</th>
<th>Strongly connected</th>
</tr>
</thead>
<tbody>
<tr>
<td>1- V(g)= e - n +2p</td>
<td>1 V(g)= e - n +p</td>
</tr>
<tr>
<td>2- V(g)= d + p</td>
<td>2 V(g)= d + p</td>
</tr>
<tr>
<td>3- V(g)= r</td>
<td>3 V(g)= r – p</td>
</tr>
</tbody>
</table>

Table I: Cyclomatic Complexity of any CFG

C. Connection Matrix:
It is a square matrix, where number of rows and number of columns are equal to number of nodes in the CFG of a program. In the simplest form when the connection exists [25], the link weight is 1, otherwise 0 (but 0 is not entered in the cell entry of matrix to reduce the complexity).

Here is the use of connection matrix in finding Cyclomatic complexity Number in figure 1 of CFG graph in figure 2

1) For each row, count the total number of 1s and write it in front of that row.
2) Subtract 1 from that count. Ignore the blank rows, if any.
3) Add the final count of each row.
4) Add 1 to the sum calculated in step 3
5) The final sum in Step 4 is the cyclomatic number of CFG.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Cyclomatic number=2+1=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3-1=2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1-1=0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1-1=0</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 1: Connection matrix of exampleCFG.

IV. BASIS PATH TESTING USING GA
A problem can be solved by Genetic algorithm if it is in the optimization form so here steps are represented to convert testing problem in to optimization problem [27]. Here the main role is of fitness function construction which converts this test data searching problem into optimization problem. Figure 3 represents steps of test data generation graphically.

A. Steps of Test Data Generation [27]:
1) CFG Construction- Control flow graph of a program under test may be constructed manually or automatically with related tools. It helps tester to select target paths.
2) Target Basis Set Selection- On the basis of cyclomatic complexity computed by connection matrix automatically and based on probability of occurrence of a specific path, target basis set is selected. Furthermore, GA can also be used as automatic tool to select target basis set.
3) Fitness Function Construction- It is used to evaluate the distance between the executed path and the target path.

4) Program Instrumentation- Here probes are inserted at the beginning of every block of the source code to monitor the program’s execution (e.g. fitness values of individuals).

5) GA Working- Initially random test data are generated from problem domain. On the basis of instrumented program and fitness evaluation GA generates new test data to achieve the target path and at last after suitable iterations on the basis of termination condition, new optimized test data is generated automatically.

Fitness Function Construction:

- **Approximation Level Based Fitness Function**: It is used to distinguish between different test data individual’s executed path from the target path by counting the number of branching nodes not traversed by current executed path, so aim is to minimize approximation level [15]. As example Fig. 4 illustrates the target path Tp which contains three decision nodes: A, B and C. If the individual path p1 diverges from the target path at the level of node A, then approximation level used for calculating the fitness function will be 2 (means p1 missed 2 decision nodes to traverse to achieve target path Tp). If the individual diverges at level of node B, then it will be 1 and if traverses all nodes then approximation level will be 0. And our aim is to minimize the approximation level for a path as fitness function.

- **Branch Distance Based Fitness Function**: It is used to distinguish between different individuals who execute the same program target path [8]. Branch distance is calculated for an individual by using branching condition in the branching node in which the target node is missed. Every branch is composed of logical expression. To force branch (to be true or false) to follow target path we have to adjust or search or optimize the input data of that branch.

Branch output depends upon input and logical expression so to get desired output from branch we places some branch distance function on the basis of logical expression in branch and required output. The branching conditions are evaluated based on a table as here table II shows an example of such a distance function [8].

Let C= (x+y>z) AND (y+z>x) AND (z+x>y) AND x>0 AND y>0 AND z>0.

Required output of C is TRUE. So, F(C)=F(C1)+F(C2)+F(C3)+F(C4)+F(C5)+F(C6), where C1=(x+y>z), C2=(y+z>x), C3=(z+x>y), C4=(x>0), C5=(y>0), C6=(z>0) and these all are also required to be TRUE.

So, F(C1)=(z-x), F(C2)=(y-x), F(C3)=(y-z), F(C4)= -x, F(C5)= -y, F(C6)= -z, Thus F(C)= -2*(x+y+z) and our aim is to minimize F(C).

<table>
<thead>
<tr>
<th>Logical expression in branch C</th>
<th>F(C)=Branch Distance If branch output=0=false</th>
<th>F(C)=Branch Distance If branch output=1=true</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=y</td>
<td>-abs(x-y)</td>
<td>abs(x-y)</td>
</tr>
<tr>
<td>x≈y</td>
<td>abs(x-y)</td>
<td>-abs(x-y)</td>
</tr>
<tr>
<td>x&gt;y</td>
<td>x-y</td>
<td>y-x</td>
</tr>
<tr>
<td>x&gt;≈y</td>
<td>y-x</td>
<td>y-x</td>
</tr>
<tr>
<td>x&lt;y</td>
<td>y-x</td>
<td>y-x</td>
</tr>
<tr>
<td>x&lt;≈y</td>
<td>x-y</td>
<td>x-y</td>
</tr>
<tr>
<td>C1 OR C2</td>
<td>F(C1)=F(C2)</td>
<td>Min ( F(C1),</td>
</tr>
</tbody>
</table>
V. CONCLUSION

In this paper, it is studied how a genetic algorithm is used to generate test data for basis path testing of a program, why basis path testing is chosen as criterion among many criteria and why GA is used among many strategies to generate test data automatically. The two important fitness functions i.e. Approximation Level based fitness function and Branch Distance based fitness function related to basis path testing are also discussed with examples. Future work will include investigating target path selection strategy, hybridization of local search methods with GA, investigating performance of GA based test data generation by using different combinations of selection, crossover and mutation operator types.

REFERENCES


