Automatic Test Case Generation for Unit Software Testing Using Genetic Algorithm and Mutation Analysis

Rijwan Khan [1], Mohd. Amjad[2]

Department of Computer Engineering, Faculty of Engineering & Technology

Jamia Millia Islamia, Jamia Nagar, New Delhi, India

rijwankhan786@gmail.com, mamjad@jmi.ac.in

***Abstract*— Software Engineers waste a lot of their time in the process of software testing. It also has been seen in the industries that a lot of money is depleting on the software process. In software testing process we apply test cases as input and check for final output. So our first concern is to choose the appropriate test cases for the software testing process. To give the correct output, it is very difficult to choose test cases. So generation of the test cases is a NP problem. To generate automatic test cases many nature inspired optimization algorithms have been used. These nature inspired optimization algorithms help to generate appropriate test cases. In this paper for generating automatic test cases genetic algorithm and mutation analysis had been used.**

Keywords*— Genetic Algorithms (GA), Mutation Analysis, Software Testing, Automatic Test Cases

I. INTRODUCTION

Developers want to produce high quality software now a days. Maintain the quality of software is a most important task. Developing these types of software, the software testing plays its own role. With the use of the software testing one can develop quality software. How much these software are reliable it is dependent on their nature of producing outputs. If output is correct it means the software is correct but before delivering the software to a customer it is most important to test it properly.

In our paper we are discussing how to generate appropriate test cases for software. For generation of appropriate test cases we are combining mutation testing with genetic algorithm. First we will generate test cases randomly and after that we will apply genetic algorithm to find optimal test case.

A. Software Testing

Software testing is process of producing correct and reliable software for the customer. It is one of the phase of the Software Development Life Cycle. (SDLC). There are many different types of software testing.

a) *Acceptance Testing*: For checking its own specific requirements, customer use this type of testing.

b) *Black Box Testing*: This type of the testing is done by the developer to check the customer requirements. Developer does this testing without knowing the internal structure of the software. He just checks require outputs.

c) *White Box Testing*: White box testing is based on knowledge of internal logic of an application code.

d) *Compatibility Testing*: This type of the testing is done by the developer to check the compatibility of the software with different operating systems, web application and hardware etc.

e) *Functional Testing*: This type of the testing is also done by the developer. He checks all validation of behavior using different inputs.

f) *Integration Testing:* In integration testing, all the coding modules are combined together for testing.

g) *Unit Testing:* To check the correct behaviour of component, developers use unit testing.

II. MUTATION TESTING

In mutation testing errors are intentionally injected into a program. The errors that are injected in the program are called mutants. There are so many errors that can be injected as mutants. Suppose we are making a program and in that program we will change operators of the program to make errors.

e.g. A program to find the area of triangle.

```
void triangle(int b, int h)
 {
Area=0.5*b*h;
 }
```

Now to introduce mutant in this program is as we change the operator '*' with operator ' +'.

```
void triangle(int b, int h)
{
    Area=0.5+b+h;
}
```

The values of b, h in the program with mutant will check how many mutants can be detected in the program. Actually mutant's analysis will check that how test cases are suitable to check the program. There are so many operators as relational operator will replace with another relational operator.

978-1-4673-8507-7/15/$31.00 ©2015 IEEE

Arithmetic operator will change with another arithmetic operator etc.

III. Genetic Algorithm

Holland published a book in 1975. The name of his book was "Adaptation in natural and artificial systems". Holland presented the idea of genetic algorithm in his book. He also described in his book that how someone can use the genetic algorithm for optimization problem. Holland proposed a theory, now this theory becomes the most powerful technique for the researchers to solve optimization problems. The principle of genetic algorithm is based on evolution and genetics. Genetic algorithm is based on "Survival of fittest". In genetic algorithm, we represent the input set by the chromosomes. These chromosomes are represented in different numbers in computer. The maximum usable representation of chromosomes is binary representation. The solution of any optimization problem using genetic algorithm is represented by the following iterative steps.

1. Initialize the population by chromosomes
2. Apply the fitness function
3. If satisfied the fitness function than stop otherwise go to step 4.
4. Select the parent for crossover or mutation
5. Generate the next generation go to step 2

There are some operation in Genetic Algorithm. These are Selection, Reproduction, and Evaluation.

Selection: In selection we have different approaches to select the parent to generate the new population. We will use any one of the following Roulette wheel or Tournament selection.

Reproduction: In reproduction crossover and mutation operations are used.

Evaluation: Evaluation is a process in which we have used our fitness function.

IV. Unit Testing

Unit testing is a type of testing in which we have different testing method. These testing are Boundary Value Testing, Equivalence Class Testing, Decision Table Based Testing, Path Testing and Control Flow Testing.

A. *Boundary Value Analysis*

This type of the testing is mainly focused on the input domain. Here the concern is about the valid input or invalid input. Basically the consideration of boundary value analysis are of different type given below.

- Normal Boundary Value Testing.
- Robust Boundary Value Testing.
- Worst Case Boundary Value Analysis and
- Robust Worst Case Boundary Value Testing.

In normal boundary value testing only valid variable value can be taken. But in robust boundary value testing we can take both valid and invalid variable values.

The tester should devise test cases to check that error messages are generated when they are appropriate, and are not falsely generated. Boundary value analysis can also be used for internal variables, such as loop control variables, indices, and pointers. Strictly speaking, these are not input variables, such as loop control variables are quite common. Robustness testing is a good choice for testing internal variables.

B. *Equivalence Class Testing*

In equivalence partitions, the partitions refers to a collection of mutually disjoint subsets, the union of which is the entire set. The idea of equivalence class testing is to identify test cases by using one element from each equivalence class. If the equivalence classes are chosen wisely, this greatly reduces the potential redundancy among test cases.

C. *Decision Table Based Testing*

A decision table has four portions: the part to the left of the bold vertical line is the stub portion; to the right is the entry portion. The part above the bold horizontal line is the condition portion, and below is the action portion.

Stub	Rule 1	Rule 2	Rule 3	Rules 4, 5
c1	T	T	T	T
c2	T	F	F	T
c3	F	---	T	F
a1	X	X		X
a2		X	X	
a3	X		X	X

Fig. 1 A Decision Table

D. *Path Testing*

In path testing, first we have to design a graph for program. With the given set of test cases for a program if all the nodes are traversed in the graph then it is called G_{node}. If all the edges traversed then it called G_{edge}. In a given graph for the given input test cases if we traverse G_{node} and G_{edge} then it is 100% coverage. In path testing we test Decision to Decision Path (DD-path) testing, simple loop coverage, predicate outcome testing, independent pairs of DD-Paths, complex loop coverage, multiple condition coverage, statistically significant coverage and all possible paths coverage.

E. *Control Flow Testing*

Control flow testing is also the type of path testing. In control flow testing we used Define/Use testing, also called All-DU path testing. In All-DU path testing first we have to define the variable and then used these variables. There are

two type of the use of the variable is available first is c-use (used in calculation) and second is p-use (used in predicate).

V. Related work

Test case generation and its automation is a key problem in software testing life cycle. If we generate automatic test cases then it can improve the efficiency of software testing and also reduce the cost of software testing [12]. As we know that simple random method is not sufficient to generate ample congeries of test data and automatic test case generation is also an optimization problem that's why we will use search based optimization techniques such as nature inspired Mata heuristics [2]. In these search based algorithms genetic algorithm is more efficient among all the other techniques [17]. Selection of suitable test data for a program using data flow information is similar to compiler optimization. Variable definition and use of variable is combine with compiler optimization [3]. An approach for inter-procedural test data generation of interactive programs has been already presented [4]. In their paper authors first find the appropriate path and then covered maximum path. They also use inter procedural control flow graph (ICFG) [4]. Different clusters for the most critical path have been identified to assign weight. This weigh is distributed among the all paths according to their critically. Initially this is taken as 100 for large programs/software and 10 for small programs/software. The weight has been divided in 80:20 ratio. The 80% weightage is given to predicate statements and 20% to the simple path [16, 17]. Test data generation is not a simple task; it is a superabundant, error prone and time consuming task. So there is need of automation of test generation process. Test cases have been developed for large system and on these test cases black box and white box testing applied [6]. A novel algorithm is proposed to support test case generation of combination design. A combination-index table (CIT) is defined, based on which the adaptive genetic algorithm (AGA) is proposed to generate test cases [14]. Process such as coding, the selection of fitness function and the improvement of hereditary operator, etc. and also generate test cases solved by genetic algorithm [22]. Genetic algorithm has been applied to search suitable solutions for that a similarity between the target path and execution path with sub path overlapped is taken as the fitness function to evaluate the fitness for individuals [23]. Some improved genetic algorithms have been applied for generating test cases automatically [24].

VI. Proposed Method

In our proposed method first we generate random test cases. Applied mutating testing to check it. If satisfied then stop.

A. *Proposed algorithm*

1. Inject the mutant in the program.
2. Generate random test cases.
3. Find the mutation score with the formula, mutation score= (number of mutants found)/ (total number of mutants).
4. If the mutant score is satisfactory (Maximum) stop, otherwise go to step 5.
5. Refine the test case using mutation score. Test case having mutation score 20% or less drop them.
6. Apply Genetic Algorithm Operations on remaining test cases to generate new test cases. Go to step 3.

B. *Genetic algorithm operations*

1. *Selection:* Select the test cases using tournament process.
2. *Mutation:* Apply the mutation operation if mutation score is less than 50 %.
3. *Crossover:* Apply crossover operation if mutation score is more than or equal to 50%.

VII. Experimental Setup

Algorithm for x^y. Where x and y are positive numbers.

```
1.  Power(x, y)
2.  If(x= =1)
3.  Return 1
4.  If(y= =1)
5.  Return x
6.  P=1, i=1
7.  While(i<=y)
8.  {P=P*x
9.  i++}
10. return P
```

Inject four mutants in this program

1. Instead of = = in 2nd step we used =.
2. Instead of = = in 4th step we used =.
3. Instead of <= in 7th step we used <.
4. Instead of * in 8th step we used +.

Now algorithm will look like this.

```
1.  Power(x, y)
2.  If(x=1)
3.  Return 1
4.  If(y=1)
5.  Return x
6.  P=1, i=1
7.  While(i<y)
8.  {P=P+x
9.  i++}
10. return P
```

Tables for the experimental values are given below.

TABLE I
CALCULATION OF X^Y

S. No	X	Y	X^Y
1	2	3	8
2	10	2	100
3	3	5	243
4	5	1	3
5	1	3	1

TABLE II
CALCULATION OF MUTATION SCORE

S. No	X	Y	Mutant 1	Mutant 2	Mutant 3	Mutant 4	Mutant score
1	2	3	NO	NO	YES	YES	50%
2	10	2	NO	NO	YES	YES	50%
3	3	5	NO	NO	YES	YES	50%
4	5	1	NO	YES	YES	YES	75%
5	1	3	YES	NO	YES	YES	75%

This table is for one suit to find the mutation score. When we applied our proposed method for 20 suits of different sizes then we covered all the mutants. After applying our proposed method 100 % mutant covered. For the genetic algorithm operations first we converted input data to 8 bits binary digits. For mutation operation we reverted one bit. i.e. 0 to one or 1 to 0 and for crossover operation we have chosen two crossover point. When we applied mutant and crossover operations then we covered 100% mutants.

TABLE III
MUTATION SCORE AFTER PROPOSED METHOD

S. No	No. of Test Suits	Size of each Suit	Mutation Score without GA (%)	Mutation Score with GA (%)
1	5	7	75	100
2	3	8	50	100
3	4	5	50	75
4	1	8	75	100
5	3	10	50	100
6	10	6	75	100
7	8	15	50	75
8	9	10	75	75
9	5	12	75	100
10	6	14	75	100

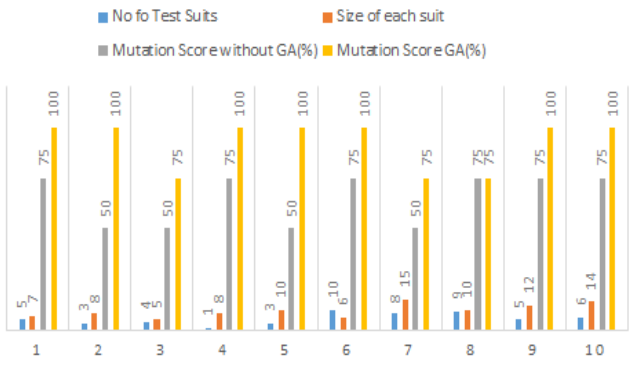

Fig. 2 Mutation Score without GA and with GA

VIII. CONCLUSION

This paper presented a new technique that uses mutation analysis with genetic algorithm to generate better automatic test cases. These new test cases are generated using GA operations mutation and Crossover. Mutation score calculated using randomly generated test cases was very low. We get improved mutation score when GA is applied to randomly generated test case.

REFERENCES

[1]. Binitha, S., and S. Siva Sathya. "A survey of bio inspired optimization algorithms." International Journal of Soft Computing and Engineering 2.2 (2012): 137-151.

[2]. Ghiduk, Ahmed S., and Moheb R. Girgis. "Using genetic algorithms and dominance concepts for generating reduced test data." Informatica 34.3 (2010).

[3]. Rapps, Sandra, and Elaine J. Weyuker. "Selecting software test data using data flow information." Software Engineering, IEEE Transactions on 4 (1985): 367-375.

[4]. Uyar, H. Turgut, A. Sima Uyar, and Emre Harmancı. "Evolutionary Software Test Data Generation Using Sequence Comparison.", international journal of principles and applications of information science and technology, December 2007, Vol 1, No1.(2007).

[5]. Hemmati, Hadi, et al. "An enhanced test case selection approach for model-based testing: an industrial case study." Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering. ACM, 2010.

[6]. Bhasin, Harsh, Neha Singla, and Shruti Sharma. "Cellular automata based test data generation." ACM SIGSOFT Software Engineering Notes 38.4 (2013): 1-7.

[7]. Haga, Hisashi, and Akihisa Suehiro. "Automatic test case generation based on genetic algorithm and mutation analysis." Control System, Computing and Engineering (ICCSCE), 2012 IEEE International Conference on. IEEE, 2012.

[8]. Badlaney, Janvi, Rohit Ghatol, and Romit Jadhwani. "An introduction to data-flow testing." Department of Computer Science, North Carolina Sate University, NCSU CSC TR-2006-22 (2006).

[9]. Oh, Jungsup, Mark Harman, and Shin Yoo. "Transition coverage testing for simulink/stateflow models using messy genetic algorithms." Proceedings of the 13th annual conference on Genetic and evolutionary computation. ACM, 2011.

[10]. Inkumsah, Kobi, and Tao Xie. "Improving structural testing of object-oriented programs via integrating evolutionary testing and symbolic execution."Automated Software Engineering, 2008. ASE 2008. 23rd IEEE/ACM International Conference on. IEEE, 2008.

[11]. Last, Mark, Shay Eyal, and Abraham Kandel. "Effective black-box testing with genetic algorithms." Lecture Notes in Computer Science 3875 (2006): 134.

[12]. Parthiban, M., and M. R. Sumalatha. "GASE-an input domain reduction and branch coverage system based on Genetic Algorithm and Symbolic Execution." Information Communication and Embedded Systems (ICICES), 2013 International Conference on. IEEE, 2013.

[13]. Nirpal, Premal B., and K. V. Kale. "Comparison of Software Test Data for Automatic Path Coverage Using Genetic Algorithm." development 4 (2011): 5.0.

[14]. Lin, Peng, et al. "Test case generation based on adaptive genetic algorithm."Quality, Reliability, Risk, Maintenance, and Safety Engineering (ICQR2MSE), 2012 International Conference on. IEEE, 2012.

[15].] Rajappa V. , Biradar A.and Panda S.(2008), Effective Software Test Case Generation Using Genetic Algorithm Based Graph Theory, First International Conference on Emerging Trends in Engineering & Technology, pp.298-303.

[16]. Srivastava, Praveen Ranjan, and Tai-hoon Kim. "Application of genetic algorithm in software testing." International Journal of software Engineering and its Applications 3.4 (2009): 87-96.

[17]. K. Koteswara Roa, G. S. V. P.Raju, Sromovasan Nagraj, Optimizing the Software Testing Efficiency by Using a Genetic Algorithm- A Design Methodology, ACM .

[18]. SIGSOFT Software Engineering Notes May 2013 Volume 38 Number 3.

[19]. Yang Cao, Chunhua Hu and Luming Li(2009), An Approach to Generate Software Test Data for a Specific Path automatically with Genetic Algorithm, International Conference on Reliability, Maintainability and Safety, pp.888-892.

[20]. S. N. Sivanandam, S. N. Deepa, Introduction to Genetic Algorithm, Springer 2008, book.

[21]. R. Khan, Mohd Amjad, Automated Test Case Generation using Nature Inspired Meta Heuristics- Genetic Algorithm: A Review Paper, International Journal of Application or Innovation in Engineering & Management (IJAIEM) Volume 3, Issue 11, November 2014 ISSN 2319 – 4847.

[22]. John H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, MI, 1975.

[23]. Wang Xibo, Su Na, Automatic Test Data Generation for Path Testing Using Genetic Algorithms, 2011 Third International Conference on Measuring Technology and Mechatronics Automation, 978-0-7695-4296-6/11 $26.00 © 2011 IEEE.

[24]. Yang Cao, Chunhua Hu and Luming Li(2009), An Approach to Generate Software Test Data for a Specific Path automatically with Genetic Algorithm, International Conference on Reliability, Maintainability and Safety, pp.888-892.

[25]. Dong, Yuehua, and Jidong Peng. "Automatic generation of software test cases based on improved genetic algorithm." Multimedia Technology (ICMT), 2011 International Conference on. IEEE, 2011.