

Comparative study on test case generation: a survey

Kulshreshtha M, Agarwal C and Kamalakannan J

School of Information Technology and Engineering, VIT University,
Vellore-632014, Tamil Nadu, India.

E-mail: chinmay.agarwal2016@vitstudent.ac.in

Abstract Testing gives means pertaining to assuring software performance. As it has been proved that the software testing phase is one of the most critical and important phases in the software development lifecycle as it lead to increase in cost, time and effort. The total aim of software industry is actually to make a certain start associated with high quality software for the end user. Software testing has quite a few underlying concerns, which are very important and need to pay attention on these issues which are effectively generating, prioritization of test cases, etc. To over come from these issues we need to pay attention and focus. In this paper we are discussing various techniques which are used to improve automated test case generation and comparing it with other existing techniques. In this we are also proposing new work which can be done in future.

1. Introduction

Testing is generally done on programming and additionally in web for testing customer and server engineering. Program testing is one of the major and essential procedures for accomplishing high quality software. Testing is accomplished to identify nearness of faults, which cause programming failures. However, programming testing is a tedious and costly task. It expends practically half of the product frame work assets required for system



development. Programming testing can likewise be characterized as procedure of confirming and validating program software to guarantee that product meets the specialized and in addition business necessities as expected in terms of technology.

Verification and validation is done to guarantee that the product meets detail and is near structural testing while approval is near the practical testing and is finished by executing software under test (SUT). Extensively, testing strategies incorporate utilitarian (black box) and basic (white box) testing. Utilitarian testing is based on practical prerequisites while structural testing is done on code itself. Black box testing is half and half of white box testing and discovery testing.

Testing should be possible either physically or naturally by utilizing testing instruments. It is found that mechanized programming testing is superior to anything manual testing. Be that as it may, not very many test information era apparatuses are economically accessible today. Different methods have been proposed for creating test information or experiments naturally. As of late, parcel of work is being accomplished for experiments era utilizing delicate processing systems like fluffy rationale, neural systems, GA, hereditary programming and developmental calculation giving keys to the issue zones of programming testing.

Developmental testing is arising procedure for naturally delivering brilliant test information. GA is notable type of the developmental calculations brought about by John Holland in United States a mid-late sixties. Developmental discovery testing is additionally connected on implanted frameworks to test its practical what's more, non-useful properties. GA has been connected in numerous advancement issues for producing test gets ready for usefulness testing, practical experiments and in numerous different ranges. GA has likewise been utilized as a part of model based experiment era. In protest arranged unit testing and in addition operating at a profit box testing, GA is utilized for programmed era of experiments. Concerning testing of web applications, numerous apparatuses, new systems and strategies have been produced to address issues like viability, testability, security, execution, correctness and unwavering quality of web application. Web applications are made out of web pages and parts and cooperation between them executes webservers, HTTP, program (the customer side) also, systems. A website page is data seen on the customer side in a solitary program window.

2. Literature Survey

Anders Hessel, Kim G.Larsen [1] describes about test cases algorithm using a tool named as UPAAL. It is a real time verification tool. The main motive of this paper writing is to propose an algorithm which the coverage of given criteria is calculated in an on the fly manner, a method which efficiently manipulates the sets of covered elements which occur during the analysis and to find extension requirement specification language which is used in UPAAL which makes it possible to explain various of coverage0 criteria. In this UPAAL is used to compute trace with less time delay which justifies a provided reachability property. In this author has extended the symbolic reachability analysis algorithm of the instrument to create traces which satisfy the simple coverage criteria that can be used to test sequences or suits to test real time system. This algorithm uses monotonically growing sets represented at bit vectors to assemble data about covered items.

Kulvinder Singh, Iqbal Kaur and Rakesh Kumar [3] are using genetic algorithm which automatically generate the test cases by using the anti-random testing technique and the produced results are verified by then versions of the module. Mutation adequacy is also used to check the completeness. The results produced using genetic algorithm with random data and hamming code are also compared and improvements have been discovered over random and hamming code testing. For generation of test code automatically two techniques are used random testing and anti-random testing. In random testing test cases are generated randomly from the described domain whereas in anti-random testing generator generates the test cases with the input of previous tests. By using two techniques i.e. random and anti-random test case generation and genetic algorithm fault propagation disperse the faulty outcome in a problem to the output and which leads to the failure of the program. All of these faults are detected and corrected. Most of the faults were distinctive to individual versions but several occurred in many version. By applying the technique of genetic algorithm with anti-random testing shows improved result over random test case generation.

In the anti-random software testing, the test cases are generated by using the hamming or Cartesian distance techniques. However, researcher used the Gray code for generating the test cases. An anti-random number generator generates the test data with use of feedback from previous tests. The tests are passed to the procedure under test, in the hope that it detects the maximum errors.

Algorithm for Anti-random Test case Generation

Consider an n bit binary number $Bin [n1:0]$ with I representing the index of the binary number.

Let $Gray [n1:0]$ be the equivalent Gray code.

1. For $I=n1$,
 $Gray [n1] = Bin [n1]$ [the most significant bit (MSB) of the Gray code is same as the MSB of original binary number.]
2. For $I=n2$ to 0 , $Gray[i] = Bin[i+1] \text{ XOR } Bin[i]$
 $[I^{\text{th}}$ bit of the Gray code is the exclusive OR (XOR) of I^{th} of the bit of the binary number and $(I+1)^{\text{th}}$ of the bit of the binary number.]

Ahmed Mateen, Marriam and Salman Afsar Awan [5] introduced a new high level test case generation process with the necessity of prioritization method to find the solution of the problem i.e. notable to identify suitable test cases with limited resources, unable to identify critical domain necessities in the test case generation process and ignore a number of generated test cases. In this author is comparing new and previous work done by other authors. In this author is using waterfall process and adding new two processes i.e. requirement prioritization and test case generation. Here requirement prioritization claims to handle large number of requirements effectively. It compares new technique with old three techniques (i.e. Heumman's method, Ryser's work and Nilawar's approach) on the basis of:

- (i) Size of the test cases
- (ii) Critical domain coverage
- (iii) Complete time.

Problems seen in this paper are

- (i) Problem in identifying and coverage of the critical domain requirements
- (ii) Size of test case
- (iii) With limited resources in efficient test case generation where new algorithm claims improve these problems.

G.J. Myers [8] provide us with a system which is capable for the optimizing of test case generation by using genetic algorithm. It also provides a better quality of process testing with in a particular time. The problems in testing product zone are generally how to gain a better and extraordinary set that is appropriate connected with cases to improve programming. Some various different systems techniques are keep on being proposed in regard of transportation consideration as the major impact of these issues. The way of the item design is measured using different methodologies and procedures. If any mix up is occurred in any

part of the under taking concluding it as essential to change the affected part of a program framework to remove the bug. It can be created just by identifying blunders and by measuring the programming nature. Thus, the way by which an item can be measured and the result is generated by the proposed theory of this methodology. This method gives a capable and proper instrument to improve the way of accessing the item. Diverse estimations are applied by the collection strategy for different sorts of various supportive frameworks. Thus, this work of investigation is completed successfully with a methodology beneficial for this proposed work.

Jasmine Minj [9] presents automatic test case generation technique in which 'Multi population genetic algorithm' is used to generate test cases. The fitness of a function is based on the multiple conditions and decision coverage criteria. MATLAB Gatool is used for implementing the algorithm for test case generation. It generates effective and efficient test cases. Test cases are optimized using 'multi population genetic algorithms'. Automatic test cases generation reduce the testing effort, time and cost. Path based test case generation is used with search based technique for which it is converted into problem of optimization. It is the process of finding the best possible solution from the feasible solution.

A benchmark triangle problem is taken as a case study. A Triangle program is given three input sides of the triangle and output will be a type of triangle according to the given input value. A triangle function is shown to check the triangle type. It is converted into control flow graph shown in Fig. The Fitness function refer to the branch distance. The four path is shown here, out of which we find the most critical path and set the parameter in MATLAB Gatool for multi population genetic algorithm.

- (1) Path: d
- (2) Path: ae
- (3) Path: abf
- (4) Path: abc

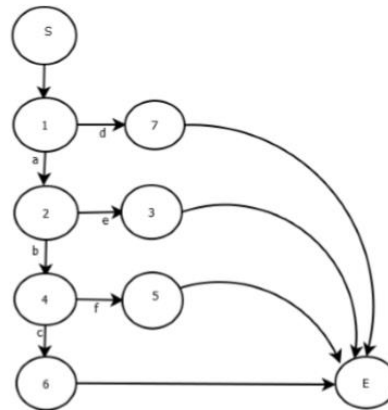


Fig. 1. Triangle problem flow control diagram

S. Desikan, R. Gopalswamy [11] provides us the knowledge about making testing easy in need to generate test cases to implement the testing process. Creating optimal test case results helps in improving the working and efficiency of the software. Search based technique is used for test case generation as testing is the major part of Software Development Lifecycle (SDLC) and is an exhausting and lengthy process as it is very difficult to handle, and generating test cases is NP hard problem. The proposed method for automatic test case generation follows a criteria of MCDC coverage guided by multi population genetic algorithm. Coverage criteria is used to measure the extent to which a given system satisfy its goal. Multiple condition decision coverage criteria is used which should satisfy as follows:

- (1) Every decision generated should execute all true as well as false criteria.
- (2) Every condition introduced must execute all true as well false criteria.
- (3) All the conditions in decisions should independently derive the output for the problem.

Using MCDC as coverage criteria we are able to evaluate software verification with good measures.

Ali, B.M.Y. and Benmaiza [12] presents a goal oriented technique for automated test data generation that uses genetic algorithm, guiding the test control dependencies within the program to search for test data, satisfying the test requirements. The genetic algorithm introduces its search generating new test data from previously obtained test data that was evaluated as good data. The algorithm provides a direction to the search by using program's control dependence graph. The execution time of lengthy search can be reduced by

implementing prototype that uses multiple process or sand load balancers that automatically prevents process or from getting locked in time consuming loops. The advantages of this approach is that it combines the features of random, goal oriented and intelligent test data generation which generates test data quickly, but providing a proper direction and focus. It also has an advantage to scale large software systems.

Roy P. Pargas, Marry Jean Harrold and Robert R. Peck [13] presents a technique for automatic test data generation that requires a generic algorithm, which is managed by the control operational dependencies with in a program used to search for test data, that satisfy the requirements. A generic algorithm is one that provides knowledge about the evolution of search varieties and provides optimal solution to the problem. In a test case generation application, the solution obtained from the generic algorithm is a test data that causes execution of given problem, data, branch, path, or definition use pair in the program under test. The test data generation method was implemented in a tool called TGen in which parallel processing was used to improve the search performance. To experiment with TGen, a random test data generator, called Radom, was also implemented. Both TGen and Random were used to test the experiment with the generation of test data for branch coverage and statement of programs. The main advantage of the approach is that it includes essential features of random, and intelligent test data generators which can quickly generate test data. The approach can also handle test data generation for problems with multiple procedures that have capabilities to work for large software systems.

Program Example:

```
integer a, b, c
read a, b, c
if (a < b)
  if (b < c)
    a = c;
  else
    c = a;
  endif
endif
print a, b, c
end Example
```

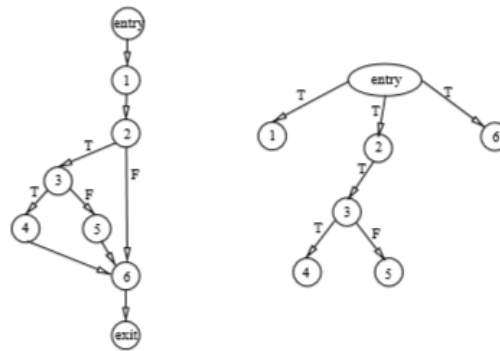


Fig. 2. Flow control graph

Itti Hooda and Chhillar, R. are discussing various methods of test case generation, minimization of test cases, selection, and evaluation & prioritization techniques. Additionally, the author has also focused on various techniques of evaluation & prioritization plus selection techniques which help the engineer's to rank and schedule test cases so that they can able to reduce or minimize the total effort, cost and time. According to the author there are two main approaches are used to automate the test cases i.e. to design test cases from the requirement and design

Specifications and second is design test cases using code. At the end, the paper gave us most important techniques of test case generation plus it also explained lifecycle phases of test cases which include minimization, selection, prioritization, and evaluation. At the end we can say that testing results depend on test execution is happened based on test cases.

Mateen A., Nazir M., & Awan S.A. [16] provide us with a structure which is having the ability to optimize the test case generation with the help of genetic algorithm (GA). It also gives better quality testing processes within time. Problem in the product test zone is typically to gain an extraordinary suitable set connected with cases to support programming. Some different systems more over techniques keep on being proposed related to transportation consideration of large number of these issues. The way of the item design is measured through different methodologies & procedures. If any error occurred in any part of the under taking workout it is essential to modify the affected part of a framework to remove the bug. It is credible just by recognizing mistakes and by computing the nature of programming. Thus the way of the item is computed and the result is motorized by the suggested theory of this investigation. This sensibility offers a proper & capable instrument to assess the way of the item. Multiple

estimations are proposed closure by the grouping of strategy for different sorts of vernaculars. Thus, this analysis of work is done successfully with a favourable methodology suggested in this work.

3. Proposed Work

- Path based test case generation is utilized with search based method for which it is converted into optimized problem. It is the procedure of finding the most ideal solution from the possible solution.
- Multi population genetic algorithm is used as a search technique that takes multiple populations.
- An application can be developed in which small size of test cases can be generated which will take less time in test case generation process and with maximum of critical domain specific requirement.
- A system can be developed with more generic language that is not limited to a predefined set of criteria as present language describes coverage area is very limited.
- We can introduce monotonic variables in the modeling language of UPPAAL. These types of variables may be useful for the specifications of other problem areas like scheduling and other planning problems.
- Here we are improving algorithm of G. J. Myers [8]

Improved algorithm

1. Introduce a variable in the program.
2. Create random test cases.
3. Find the change score with the formula,
change score= (number of changes found) / (total number of changes).
4. If the change score is acceptable (Maximum) then stop, else go to step 5.
5. Refine the algorithm using change score. Obtain test case having change score 20%, if less then drop the values.
6. Now use Genetic Algorithm on the remaining test cases to deliver new tests. Then go to step 3.

Algorithm for xy, where x and y are positive integers.

1. Power(x, y)
2. If(x==1)
3. Return 1
4. If(y==1)
5. Return x
6. A=1, i=1

```
7. While(i<=y)
8. {A=A*x
9. i++}
10. return A
```

Insert four different values in the program

Now the algorithm looks like

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

According to the optimized flow of algorithm, we find the number of changes by using the algorithm. First, some values are added in. Change values are some errors, which are generated to find the optimal test case. Then, evaluate the performance of test case and then first step is initialized. The next step is evaluating the change values found. If the number is less than the least number of changes then it means test case failed to find errors. Else if the condition is no, then we calculate the fitness by finding the exact numbers of changes. Next if change values are found to be more than 50%, then it displays best fit but if the found values are less than 50% changes then it displays change number's value. If the total number of changes found in the 2nd step then no more steps are needed to find errors. Thus the test case provides the optimal solution by finding all the errors.

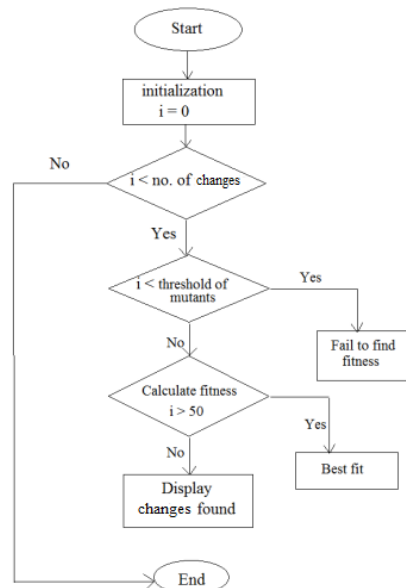


Fig. 3. The structure of algorithm

4. Conclusion

In this paper, applications of Genetic Algorithm in different types of program testing are discussed. It is found that by using Genetic Algorithm, the performance and the outcomes of testing can be highly improved. Our future work will include applying Genetic Algorithm for regression testing in web based applications. In future, we plan to use Genetic Algorithm along with other soft computing techniques like neural networks or fuzzy logics used for test case generation. We additionally plan to use Genetic Algorithm in integration testing for finding optimal test order.

We use various test cases technique specifically from UML diagram, where the design is reused. By utilizing our approach defects in the design model can be detected during the analysis of the model itself. So, the deformities can be evacuated as early as possible, thus lessening the cost of deformity removal. First we generate test case generation using model based testing, random based testing, genetic algorithm and test scenarios from the activity diagram and then for each scenario the corresponding sequence diagram should be generated. After analysing and investigating each category, its critical values and constraints are produced and particular test cases are determined. Test coverage criteria achieved is another advantage of our approach.

References

- [1] Hessel A, Larsen K G, Nielsen B, Pettersson P and Skou A 2003 Time-Optimal real-time test case generation using UPPAAL *International Workshop on Formal Approaches to Software Testing* 114-130
- [2] Applegate D and Cook W 1991 A computational study of the job-shop scheduling problem *ORSA Journal on computing* **3** 149-156
- [3] Singh K, Kaur I and Kumar R 2012 Automatic Test Case Generation using Genetic Algorithm with Antirandom Population *IJACE* **5**(1) 21-27
- [4] Behrmann G, Fehnker A, Hune T, Larsen K, Pettersson P and Romijn J 2001 Efficient guiding towards cost-optimality in uppaal *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* 174-188.
- [5] Mateen A, Nazir M and Awan S A 2016 Optimization of Test Case Generation using Genetic Algorithm (GA) *International Journal of Computer Applications Foundation of Computer Science* **151**(7) 9
- [6] Dill D 1990 Timing assumptions and verification of finite-state concurrent systems In *Automatic verification methods for finite state systems* 197-212 Springer Berlin/Heidelberg
- [7] Frankl P G and Weyuker E J 1988 An applicable family of data flow testing criteria *IEEE Transactions on Software Engineering* **14** 1483-1498
- [8] Myers G J, Sandler Cand Badgett T 2011 the art of software testing John Wiley & Sons
- [9] Minj J 2013 Feasible Test Case Generation using Search based Technique. *International Journal of Computer Applications* **70**(28) 51-55
- [10] Hetzel W Cand Hetzel B 1991 The complete guide to software testing. John Wiley & Sons, Inc.
- [11] Desikan S 2006 Software testing: principles and practice Pearson Education India
- [12] Ali Y M B and Benmaiza F 2012 Generating test case for object-oriented software using genetic algorithm and mutation testing method *International Journal of Applied Metaheuristic Computing* **3**(1) 15-23
- [13] Pargas R P, Harrold M J and Peck R R 1999 Test-data generation using genetic algorithms *Software Testing Verification and Reliability* **9**(4) 263-282
- [14] Hooda I and Chhillar R 2014 A review: Study of test case generation techniques *International Journal of Computer Applications* **107**(16) 33-37
- [15] Singh A, Garg Nand Saini T. 2014 A hybrid Approach of Genetic Algorithm and Particle Swarm Technique to Software Test Case Generation *International Journal of Innovations in Engineering and Technology* **3**(4) 208-214