DEVELOPMENT OF GENETIC ALGORITHMS TO IMPROVE TEST CASE GENERATION IN SOFTWARE ENGINEERING

Rajat Bhatia*, Prof Boominathan P, Prof Ramani S
Software Engineer, Reliance Retail (JIO), Bangalore, India.
School of Computing Science and Engineering, VIT University, Vellore, India.

Email: rajat.bhatia@yahoo.com

Received on 22-07-2016 Accepted on 26-08-2016

Abstract:
Automation plays a major role for improving the test efficiency of software test organization in software engineering. Projects use more people for manual testing or use automation tools or techniques to improve the level of test automation for ensuring test coverage satisfaction and thereby reducing risk. The choice of selection depends on reduction in project cycle time or reduction in time for test. This paper discusses about software automation steps and explains an algorithm for addressing the issues faced in system management automation of various services like DHCP, DNS, and Firewall etc.

Keywords: Algorithmic solution, Automation design, Frame work, Hash values, Test automation.

1. Introduction:

1.1. Software testing

Software testing is a process of executing a program or application with the intent of finding the software bugs. It can be stated as the process of validating and verifying that a software program or application or product. It meets the business and technical requirements that guided its design and development. It can be implemented with the same characteristic. It is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. It is laborious, costly and time consuming task: it spends almost 50% of software system development resources. Software testing is performed for defect detection and reliability estimation the goal of which is to design a set of minimal number of test cases and test data such that it reveals as many faults as possible.

In software testing process each test case has an identity and it is associated with a set of inputs and a list of outputs. Test data generation is how to identify a set of program input data that satisfies a given testing criterion. Testing is a static and dynamic activity. Black box and white box testing are two categories of software testing which includes...
unit, interface, functionality, performance, usability, security and operability testing of software. It is executed by conducting a program developed with test inputs and comparing the observed output with the expected one. It is difficult to test the whole software, therefore selective parts of the software are considered for testing. As the input space of the Software Under Test (SUT) might be very large, testing has to be conducted with a representative subset of test cases. Creating relevant subset of test cases during software testing is the most critical activity. During testing the quality and the significance of the overall test are directly affected by the set of test cases that are used. An effective set of test cases is one that has a high probability of detecting faults. This requirement of effective test cases demands the generation of ‘Good’ automated test cases. Test data can be used to create test cases. Another critical task in automation of software testing is generation of test data to satisfy a given adequacy criterion. There is need to explore these aspects of test data generation in order to increase the degree of automation and efficiency of software testing.

1.2. Genetic algorithm

According to Aljahdali S.H[1], Genetic Algorithms (GAs) are adaptive heuristic search techniques premised on the evolutionary ideas of genetic and natural selection. The concept of GAs was designed to simulate processes in evolution system. Genetic algorithms represent an intelligent exploitation of a random search within a defined search space to solve a problem. GAs have been widely studied, experimented and applied in many fields in the engineering worlds. Chen Yong[3] says they are based on the principles of the evolution via natural selection, employing a population of individuals that undergo selection in the presence of variation-inducing operators, such as mutation and recombination. GA’s are useful and work efficiently when the search space is large, complex and poorly understood, when domain knowledge is scarce or expert knowledge is difficult to encode. It’s also useful when there is a need to narrow the search space and in case of failure of traditional search methods, says Bouchachia A(2007)[2]. The genetic algorithm loops over an iteration process to make the population evolve. In each iteration we have the following steps:

Step 1: Initialize population;
Step 2: Evaluate population;
Step 3: While Termination Criteria Not Satisfied repeat step 4 to 6;
Step 4: Select parents for reproduction;
Step 5: Perform recombination and mutation;
Step 6: Evaluate population;
1.2.1. Simple genetic algorithm

A simple genetic algorithm is given below for an example:

```{initialize population; evaluate population; while TerminationCriteriaNotSatisfied { select parents for reproduction; perform recombination and mutation; evaluate population; }}
```

Genetic Algorithms always begin with a set of individuals as the first generation, which are then sampled at random from the problem domain. All the algorithms are developed to perform a series of operations which transforms the present generation into a newer and fitter generation. Each and every individual in each generation is evaluated with a fitness function. Aljahdali S.H[1] stated based on this evaluation, the evolution of the individuals approach the optimal solution. The most common operations of genetic algorithms are then designed to produce the most efficient solution for the target problem. These primary operations include:

a) Reproduction: The reproduction probability to each individual is assigned based on the output of the fitness function. The one with a greater probability for reproduction is given a higher ranking. As a result, the fitter individuals are allowed a better survival chance from one generation to the next.

b) Crossover: This operation is used to produce the descendants that make up the next generation. This operation involves the following crossbreeding procedures:

i) Select two individuals randomly as a couple from the parent generation.

ii) Then select a position of the genes, corresponding to this couple, as the crossover point. Thus, each gene is divided into two parts.

iii) Exchange the first parts of both the genes corresponding to the couple.

iv) Add the two resulted individuals to the next generation.
c) Mutation: The operation picks up a gene at random and changes its state according to the mutation probability. The mutation probability is given intuitively since there is no definite way to determine the mutation probability. Upon completion of crossover processing and mutation operations, there will be an original parent population and a new offspring population. A fitness function should be devised to determine which of these parents and offsprings can be survived into the next generation. After performing the fitness function, these parents and offsprings are filtered and a new generation is formed. These operations are iterated until the expected goal is achieved. Genetic algorithms guarantee high probability of improving the quality of the individuals over several generations.

1.3. Genetic Algorithm in Software Testing

The key problem in software testing and its automation are the test cases and test data generation improves the efficiency and effectiveness and also lowers the high cost of testing. Using random, symbolic and dynamic approach for the generation of test data is not enough to generate adequate amount of test data. Many other problems, like the inefficiency to generate test data for complex programs and the non-recognition of occurrences of infinite loops makes all these techniques unsuitable for generating test data says Ghiduk[4]. Therefore, there is need to generate test data using search based technique. There is also a need of generating these test cases that can concentrate on error prone areas of code. The genetic algorithm is applied for automating test cases for the functional testing, where the program is treated as a black box and then extracted from the code the necessary information to calculate the objective function and then measuring the time it takes in the process. Genetic algorithmis used to automatically generate test cases that provides a good coverage in terms of the path it tests or visits within the user interface of tested application. It is also used to convert the task of test case generation into an optimal problem. One of the genetic algorithm applications in software testing is generation of suitable test data[3]. The application of GA in Software Testing is an emerging area of research which brings about the cross fertilization of ideas across two domains. It is also used to generate all the test cases while ensuring the generated test cases that are not redundant. This maximizes the test coverage for the generated test cases. In order to carry out the effectiveness of the test cases and test data the quantification, measurement and the perfect modeling is required. This is done by using the appropriate suite of software test metrics. The test metrics are then used to measure the number, complexity, quality and coverage of generated test cases and test data to make the testing more reliable and quantifiable.

2. Objectives:

The objectives of the present research are to identify and analyze the software testing parameters and to study the existing automated test data and test cases used for software testing. It is also to analyze the Genetic Algorithm for
improving/developing test data and test cases. Also the basic nature of existing genetic transformations used for software testing and its applications for generating the efficient software test cases can be studied while this research.

It is also to improve upon the existing test data and test cases or generate new ones for software testing, using suitable genetic algorithm operators and to study and improvise the existing test metrics for measuring the number, quality, complexity and coverage of generated test data and test cases. It is also to compare the efficiency and effectiveness of the improvised / the newly developed methods with the existing software testing approaches.

3. Methodology:
In order to achieve the above mentioned objectives the following methodology is followed. The parameters are identified to improve the software testing methods on the basis of their dynamic and static nature and then the factors of genetic algorithm responsible for developing efficient test data and test cases are explored. According to Ghiduk [4] the new / refine existing genetic algorithm approach is then designed to improve testing methods which is then evaluated for the effectiveness of such approach on actual case studies. Then evaluate the efficiency of generated test data and test cases using test metrics which is then refined for the existing / developing of new test metrics to make testing more effective. The performance and effectiveness of the newly designed approach with the existing ones is then compared. On the basis of comparison the conclusions will be drawn and presentation of new results in the form of papers and thesis.

![Flow Chart](image)

**Figure 1: Mutation based test case generation.**

### 3.1. Flow Chart

![Flow Chart](image)
4. Tools and techniques:

The automation of test data and test cases will be done with the help of Genetic Algorithm approach. Open source automated testing tools will be used to create, manage and run tests for testing types like functional testing, unit testing, coverage testing and object oriented testing says Haichang Gao[5]. Other tools needed as per requirement of the proposed research, may be used from time to time. Control Dependency graphs are defined in terms of the program’s control flow graph and the post dominance relation that exists among the nodes in the control flow graph.

In a Control flow graph, nodes represent statements and edges represent the flow of control between statements- an edge (X,Y) in a graph represents the flow of program from X to Y. Below is a Control Flow Graph augmented with entry and exit nodes. For statements X and Y in a CFG, Y is control dependent on X if and only if there exists a directed path P from X to Y with all Z in P(excluding X and Y) post dominated by Y and X is not post dominated by Y. In a CDG, nodes represent statements and edges represent the control dependencies between statements- an edge (X, Y) in a CDG represents Y being dependent on X.

4.1. Example of a program

integer i, j, k

1. read i, j, k
2. if (i<j)
3. if(j<k)
4. i=k;
   else
5. k=i;
6. print i, j, k

end

4.2. Control Flow Graph

![Control Flow Chart](image)

Figure 3: Control Flow Chart.

4.3. Control Dependent Graph

![Control Dependent Chart](image)

Figure 4: Control Dependent Chart.

5. Results:

Test data generation is quite labor-intensive and expensive in software testing. The automation of test process may result in significant reduction in the cost of software development. The automation of test cases that concentrate on the error prone areas will help in improving the quality of the software. Test data is often generated manually as such demand for automatic test data generation is high. There has also been an increase in the demand for coverage assessment tools elsewhere in the software development community. This increases the demand for test data generation tools. Genetic algorithms are one of the best ways to solve a problem about which only a little knowledge is available. Genetic algorithms use the principles of selection and evolution to produce several solutions to a given
problem. Genetic algorithms tend to thrive in an environment in which there is a very large set of candidate solutions and in which the search space is uneven and has many hills and valleys.

6. Conclusions:

As we have seen that a number of optimization techniques have been applied for the test case generation but none could achieve the best performance for every piece of code. Since, the test case generation has become an optimization problem hence scoperemains open to apply some more techniques to achieve better results[7]. In this paper, through discussion about GA as optimization techniques for test case generation is covered, which will pave the path for further work in this direction.

7. References:


Corresponding Author:
Rajat Bhatia,
Email: rajat.bhatia@yahoo.com