Software Testing using Evolutionary Approach

Naveen Singh[*], Mrs. Kavita Agarwal[**]

[*] Software Testing Engineer
[**] Professor, Integral University Lucknow (India)

Abstract- Software testing is one of the very essential phase of the SDLC .in this phase we have to generate lots of test cases for testing applications. As principal says 'exhaustive testing' is not possible, here we can use the genetic algorithm to reduce the number of test cases as well as select best test cases that can give effective result. Genetic algorithm is based on 'fittest of survival' concept so here we select fittest test case that can give appropriate result.

Index Terms- Testing, GA

I. Introduction

The verification and validation of software through dynamic testing is an area of software engineering where progress towards automation has been slow. In particular the automation design and generation of test data remains, by and large, a manual activity. Software testing remains the primary technique used to gain consumers' confidence in the software. The process of testing any software system is an enormous task which is time consuming and costly . Software testing is laborious and time-consuming work; it spends almost 50% of software system development resources Generally, the goal of software testing is to design a set of minimal number of test cases such that it reveals as many faults as possible. As mentioned earlier, software testing is a lengthy and time-consuming work . Absolutely, an automated software testing can significantly reduce the cost of developing software. Other benefits include: the test preparation can be done in advance, the test runs would be considerably fast, and the confidence of the testing result can be increased. However, software testing automation is not a straight forward process. For years, many researchers have proposed different methods to generate test data automatically, i.e. different methods for developing test data/case generators . The development of techniques that will also support the automation of software testing will result in significant cost savings. The application of artificial intelligence (AI) techniques in Software Engineering (SE) is an emerging area of research that brings about the cross fertilization of ideas across two domains. A number of researchers did the work on software testing using artificial inelegance; they examine the effective use of AI for SE related activities which are inherently knowledge intensive and human-centered. These issues necessitate the need to investigate the suitability of search algorithms, e.g. simulated annealing, genetic algorithms, and ant colony optimization as a better alternative for developing test data generators Using evolutionary computations, researchers have done some work in developing genetic algorithms (GA)-based test data generators . A variety of techniques for test data generation have been developed previously and these can be categorized as structural and functional testing.

In this paper, we present the results of our research into the application of GA search approach, to identify the most appropriate test case for testing.

Testing is one of the most used software quality assessment methods. There are two important processes when testing object oriented software are used. First, the software has to be initialized with a set of values. These values are used to set a number of variables that are relevant for the test case. The values of these variables define a single state from the possible set of states, the software can be determined. These values can either be a primitive value such as an integer or complex values such as an object. With the software testing initialized, its method takes one or more software specification, defines the output of the software and what a valid input is. Since the number of more objects as parameters, these objects also have to be initialized. To determine if the test case passed or fail, a software specification has to be used. The number of possible states software may have is exponential, it is impossible to test all of them. Software testing , is one of the major and primary techniques for achieving high quality software. Software testing is done to detect presence of faults [, which cause software failure. However, software testing is a time consuming and expensive task.

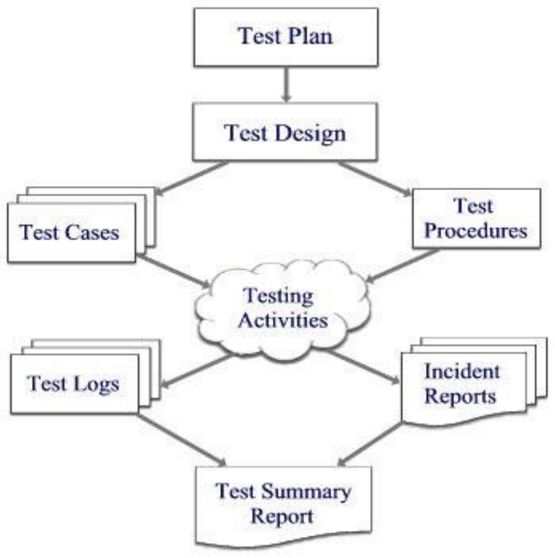

Figure 1 shows all the activities of software testing according to IEEE829.

II. Testing techniques

2.1 Black box testing:

In black box testing, the internal structure and behavior of the program under test is not considered. The objective is to find out solely when the input-output behavior of the program does not agree with its specification. In this approach, test data for software are constructed from its specification, Beizer [1990], Ince [1987] and Frankl and Weiss [1993]. The strength of black box testing is that tests can be derived early in the development cycle. This can detect *missing logic* faults mentioned by Hamlet [1987].

The software is treated as a black box and its functionality is tested by providing it with various combinations of input test data. Black box testing is also called *functional* or *specification based testing*. In contrast to this is white box testing.

2.2 White box testing:

In *white box* testing, the internal structure and behaviour of the program under test is considered. The structure of the software is examined by execution of the code. Test data are derived from the program's logic. This is also called *program-based* or *structural testing*, Roper [1994]. This method gives feedback e.g. on coverage of the software.

There are several white box (structural) testing criteria:

• *Statement Testing*: Every statement in the software under test has to be executed at least once during testing. A more extensive and stronger strategy is branch testing.

• *Branch testing*: Branch coverage is a stronger criterion than statement coverage. It requires every possible outcome of all decisions to be exercised at least once Huang [1975], i.e. each possible transfer of control in the program be exercised. This means that all control transfers are executed, Jin [1995]. It includes statement coverage since every statement is executed if every branch in a program is exercised once. However, some errors can only be detected if the statements and branches are executed in a certain order, which leads to path testing.

• *Path testing*: In path testing every possible path in the software under test is executed; this increases the probability of error detection and is a stronger method than both statement and branch testing. A path through software can be described as the conjunction of predicates in relation to the software's input variables. However, path testing is generally considered impractical because a program with loop statements can have an infinite number of paths. A path is said to be '*feasible*', when there exists an input for which the path is traversed during program execution, otherwise the path is unfeasible.

III. Genetic Algorithm

Genetic Algorithms have been introduced in the sixties by Professor John Holland at university of Michigan as models of an Artificial Evolution . In the thirty past years, they have been successfully applied to a wide range of problems such as Natural Systems Modeling (e.g. Artificial Life environments, immune system modeling , Machine Learning systems, and optimization. GAs handle a population of individual (chromosomes) often modeled by vector of binary genes. Each one encodes a potential solution to the problem and so-called fitness value, which is directly correlated to how good it is to solve the problem. In general, the basic approaches are to test software consists of using formal specifications to design an application. This approach is very strict but unfortunately not often used because the breadth of formal specification methods does not encompass all the functionality needed in today's complex applications.

The second approach consists of doing test as part of the traditional engineering models (e.g. waterfall, spiral, prototyping) that have a specific phase for testing generally occurring after the application has been implemented.

The Genetic Algorithms are the heuristic search and optimization techniques that mimic the process of natural evolution." Select The Best, Discard The Rest".

Ex. Giraffes have long necks

1-Giraffes with slightly longer necks could feed on leaves of higher branches when all lower ones had been eaten off.

2- They had a better chance of survival.

3-Favorable characteristic propagated through generations of giraffes.

4-Now, evolved species has long necks.

Fig-2

This longer necks may have due to the effect of mutation initially. However as it was favorable, this was propagated over the generations.

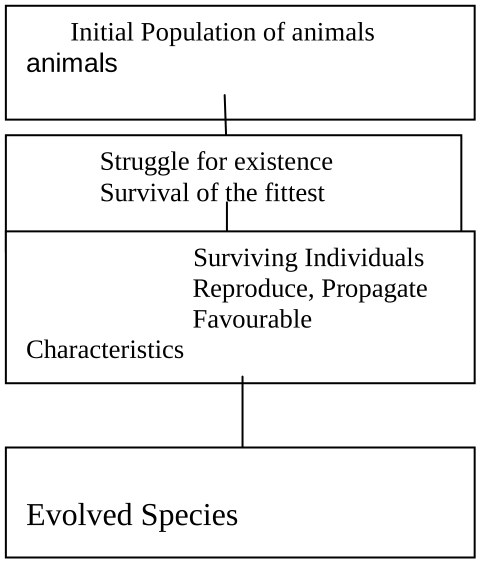
Initial Population of animals animals
Struggle for existence
Survival of the fittest
Surviving Individuals Reproduce, Propagate Favourable Characteristics
Evolved Species

Fig-3 Evolution of Species

Thus genetic algorithms implement the optimization strategies by simulating evolution of species through natural selection

3.1 Biological Background:

All living organisms consist of cells. In each cell there is the same set of **chromosomes**. Chromosomes are strings of DNA and serves as a model for the whole organism. A chromosome consist of **genes**, blocks of DNA. Each gene encodes a particular protein. Basically can be said, that each gene encodes a **trait**, for example color of eyes. Possible settings for a trait (e.g. blue, brown) are called **alleles**. Each gene has its own position in the chromosome. This position is called **locus**. Complete set of genetic material (all chromosomes) is called **genome**. Particular set of genes in genome is called **genotype**. The genotype is with later development after birth base for the organism's **phenotype**, its physical and mental characteristics, such as eye colour, intelligence etc. During reproduction, first occurs **recombination** (or **crossover**). Genes from parents form in some way the whole new chromosome. The new created offspring can then be mutated. **Mutation** means, that the elements of DNA are a bit changed. This changes are mainly caused by errors in copying genes from parents. The **fitness** of an organism is measured by success of the organism in its life.

3.2 Search Space:

If we are solving some problem, we are usually looking for some solution, which will be the best among others. The space of all feasible solutions (it means objects among those the desired solution is) is called **search space** (also state space). Each point in the search space represent one feasible solution. Each feasible solution can be "marked" by its value or fitness for the problem. We are looking for our solution, which is one point (or more) among feasible solutions - that is one point in the search space. The looking for a solution is then equal to a looking for some extreme (minimum or maximum) in the search space. The search space can be whole known by the time of solving a problem, but usually we know only a few points from it and we are generating other points as the process of finding solution continues.

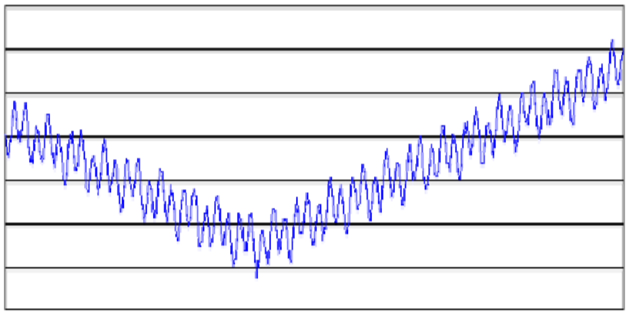

Fig-3 search Space

The problem is that the search can be very complicated. One does not know where to look for the solution and where to start. There are many methods, how to find some **suitable solution** (ie. not necessarily the **best solution**), for example **hill climbing**, **tabu search**, **simulated annealing** and **genetic algorithm**. The solution found by this methods is often considered as a good solution, because it is not often possible to prove what is the real optimum.

3.3 Basic Description:

Genetic algorithms are inspired by Darwin's theory about evolution. Solution to a problem solved by genetic algorithms is evolved. Algorithm is started with a **set of solutions** (represented by **chromosomes**) called **population**. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. Solutions which are selected to form new solutions (**offspring**) are selected according to their fitness - the more suitable they are the more chances they have to reproduce. This is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied.

3.3.1 Outline of the Basic Genetic Algorithm :

1. **[Start]** Generate random population of *n* chromosomes (suitable solutions for the problem)
2. **[Fitness]** Evaluate the fitness *f(x)* of each chromosome *x* in the population
3. **[New population]** Create a new population by repeating following steps until the new population is complete
    1. **[Selection]** Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
    2. **[Crossover]** With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
    3. **[Mutation]** With a mutation probability mutate new offspring at each locus (position in chromosome).
    4. **[Accepting]** Place new offspring in a new population
4. **[Replace]** Use new generated population for a further run of algorithm

5. **[Test]** If the end condition is satisfied, **stop**, and return the best solution in current population
6. **[Loop]** Go to step **2**

3.3.2 Operators of GA:

As you can see from the genetic algorithm outline, the crossover and mutation are the most important part of the genetic algorithm. The performance is influenced mainly by these two operators. Before we can explain more about crossover and mutation, some information about chromosomes will be given.

Encoding of a Chromosome:

The chromosome should in some way contain information about solution which it represents. The most used way of encoding is a binary string. The chromosome then could look like this:

Chromosome 1	1101100100110110
Chromosome 2	1101111000011110

Each chromosome has one binary string. Each bit in this string can represent some characteristic of the solution. Or the whole string can represent a number - this has been used in the basic GA applet.

Of course, there are many other ways of encoding. This depends mainly on the solved problem. For example, one can encode directly integer or real numbers; sometimes it is useful to encode some permutations and so on.

Crossover:

After we have decided what encoding we will use, we can make a step to crossover. Crossover selects genes from parent chromosomes and creates a new offspring. The simplest way how to do this is to choose randomly some crossover point and everything before this point point copy from a first parent and then everything after a crossover point copy from the second parent.

Crossover can then look like this (| is the crossover point):

Chromosome 1	11011 | 00100110110
Chromosome 2	11011 | 11000011110
Offspring 1	11011 | 11000011110
Offspring 2	11011 | 00100110110

There are other ways how to make crossover, for example we can choose more crossover points. Crossover can be rather complicated and very depends on encoding of the encoding of chromosome. Specific crossover made for a specific problem can improve performance of the genetic algorithm.

Mutation:

After a crossover is performed, mutation take place. This is to prevent falling all solutions in population into a local optimum of solved problem. Mutation changes randomly the new offspring. For binary encoding we can switch a few randomly chosen bits from 1 to 0 or from 0 to 1. Mutation can then be following.

Original offspring 1	1101111000011110
Original offspring 2	1101100100110110
Mutated offspring 1	1100111000011110
Mutated offspring 2	1101101100110110

The mutation depends on the encoding as well as the crossover. For example when we are encoding permutations, mutation could be exchanging two genes.

Crossover and Mutation Probability:

There are two basic parameters of GA - crossover probability and mutation probability.

Crossover probability says how often will be crossover performed. If there is no crossover, offspring is exact copy of parents. If there is a crossover, offspring is made from parts of parents' chromosome. If crossover probability is **100%**, then all offspring is made by crossover. If it is **0%**, whole new generation is made from exact copies of chromosomes from old population (but this does not mean that the new generation is the same!).

Crossover is made in hope that new chromosomes will have good parts of old chromosomes and maybe the new chromosomes will be better. However it is good to leave some part of population survive to next generation.

Mutation probability says how often will be parts of chromosome mutated. If there is no mutation, offspring is taken after crossover (or copy) without any change. If mutation is performed, part of chromosome is changed. If mutation probability is **100%**, whole chromosome is changed, if it is **0%**, nothing is changed.

Mutation is made to prevent falling GA into local extreme, but it should not occur very often, because then GA will in fact change to **random search**.

IV. Application of GA in Software Testing

We proposed in this paper application of genetic algorithm in testing of software application. when we test any application or any program we can use three things to generate fitness function. GA helps in selecting the various test cases whose fitness function is based on these factors.

- Likelihood
- Close to boundary value
- Branch coverage

These factors contribute to the selection of the test suit. These factors here are used as the evaluation of the test suit for its goodness. These factors can be used as the fitness function of the GA to find the optimal solution i.e. the best set of test cases. These factors evaluate the goodness of a test suit (set of test cases). A good test case has more chances of finding a bug in a program.

4.1 Likelihood:

The paths, which are more likely to be executed then others should be given higher priority for testing. *Likelihood* of any test suit is higher than the likelihood of any other test suit, if the test cases of the test suit follow the paths that are more likely to be executed. The likelihood factor will contribute to the selection of any particular test suite for execution. More the likelihood of a test suite higher is its probability to be selected for executions.

4.2 Close to boundary value:

As the chances of bugs are mostly at the boundary values, so the test cases close to boundary values must be given higher priority than the other ones for testing. *Close to boundary value* is the factor that represents the 'how much the test case values are closer to the boundaries'.

4.3 Branch coverage:

Most of the earlier automated test tools use the branch coverage criterion for selecting test cases. Branch coverage means the percentage of no of edges/branches of the control flow graph covered by the test suit. *Control flowgraph* is graphical notation of the program that shows the flow of control of that program.

V. Conclusion

Genetic Algorithms can be applied n software testing, and . Software testing is also an optimization problem with the objective that the efforts consumed should be minimized and the number of faults detected should be maximized. Software testing is considered most effort consuming activity in the software testing. Although a number of testing techniques and adequacy criteria have been suggested in the literature but it has been observed that no technique/criteria is sufficient enough to ensure the delivery of fault free software consequential to the need of automatic test case generation to minimize the cost of testing. The simulation shows that the proposed GAs with the specification can find solutions with better quality in shorter time. The developer uses this information to search, locate, and segregate the faults that caused the failures. While each of these areas for future consideration could be further investigated with respect to applicability for software testing, as demonstrated by the examples of this paper, the simple genetic algorithm approach presented in this paper provides in itself a useful contribution to the selection of test cases and a focused examination of test results.

Acknowledgments

This work is supported by Mobile QA Zone (An Application Testing Community), resources from Prof. Dr. M. Rizwan Beg, Head of department of computer science and engineering, Integral University, Lucknow. They would also thank the anonymous reviewers for their significant and constructive critiques and suggestions, which substantially improved the quality of this paper.

References

[1] Application of Genetic Algorithm in Software Testing Praveen Ranjan Srivastava1 and Tai-hoon Kim2.

[2] Software Testing using Genetic Algorithm 1Sanjay Kumar Sonkar, 2Dr. Anil Kumar Malviya, 3Dharmendra Lal Gupta, 4Ganesh Chandra.

[3] Introduction To Genetic Algorithms, Dr. Rajib Kumar Bhattacharjya Department of Civil Engineering IIT Guwahati

[4] http://www.obitko.com/tutorials/genetic-algorithms

[5] Optimization of Functional Testing using Genetic Algorithms Kulvinder Singh and Rakesh Kumar

Authors

First Author – Naveen Singh, Software Testing Engineer
Second Author – Mrs. Kavita Agarwal, Professor, Integral University Lucknow (India)