Optimization of Test Data in Mutation Testing Method by Using Improved Genetic Algorithm

Mohsen Fallah Rad, Sajad Bahrekazemi, Shahram moadab
Department of Software Engineering, College of Engineering, Lahijan Branch, Islamic Azad University, Lahijan, Iran.
mfalahrad@gmail.com

M.Sc Student software engineering, Department of Computer Engineering, University of Guilan, Rasht, Iran and member of Young Researchers Club, Langeroud Branch, Islamic Azad University, Langeroud, Iran.
Shahrekazemi@ymail.com

Software Engineering Department, Islamic Azad University, Qazvin Branch Qazvin, Iran.
shm7_7@yahoo.com

Abstract

In this paper, a method for the optimization of data test in mutation testing is introduced. Mutation testing tries to introduce faults to the program under test and to find them by test data sets, if they are not found these failures, testers should improve this data. During this process, program is analyzed. Usually in this kind of testing method, evolutionary algorithms, such as Simple Genetic and Bacteriological are used for improving of test data set. Measurement of mutation rate and crossover are considered a big challenge, generally they are obtained by of repeated experiments with a large degree of difficulty. In the suggested algorithm, however, this task is done automatically.

Keywords: Software Testing, Mutation Testing, Genetic Algorithm, Bacteriological Algorithm, Improved Genetic Algorithm.

I. Introduction

Testing of software is a very expensive process. Around 30 to 40 percent of the software company budget is specified to find the errors before offering the software to the markets (G. J. Myers, 1979). The aim of the test is to detect the errors as far as possible using the test data. Testing is one of the techniques of a programmer to verify the validity and correctness of a program. It’d be expensive to create and develop the test data manually. Automation of the data development techniques can be used in order to facilitate this process. There are generally two types of test: Black-Box Test and White-Box Test. In Black-Box Test, the internal structure and the input-output behavior of the system, and in White-Box Test, the program structure and code are considered. In this article, a system of mutation tests, in which
the test data development is operated by Genetic Algorithm, is presented. The Mutation Test is considered as the White-Box Test. The development of the test data is the most important and the most expensive section of a Mutation Testing. Today, according to the development of the Genetic Algorithm and their rapidity in searching for an optimum solution, they are very common to be used. In Genetic Algorithm, the calculation of the mutation rate and the cross over are always a challenge to the examiner. Finding it requires different tests. In this article, it’s intended to introduce and modify an algorithm which can determine these rates automatically.

II. Mutation Test In Brief

In Mutation Test, the errors are introduced by small modifications in the main program code. The Mutation Test was first introduced by "Hamlet" (M. R. Woodwar et al., 1988), (G. J. Myers, 1979). These modified programs are named Mutants. If the output of the main program and the Mutant varied, it’d be called a Killed Mutant. A mutant is called Alive, when no data test can eliminate it. Surviving for a mutant means that data test is insufficient to detect the errors. In this situation, the data test has to be modified as far as these Mutants are detected. In some occasions, no sample of test data can find a difference between a Mutant and the main program, where in this case they are called the Equivalent Mutant.

For instance, consider the program P:

\[\text{if (c==a +b)}\]
\[\text{do This();}\]
\[\text{else do That();}\]

Some mutants, made according to this program are as follows:

- P1: if (c==a -b)
  do This();
  else do That();

- P2: if (c==a *b)
  do This();
  else do That();

- P3: if (c==a /b)
  do This();
  else do That();

- P4: if (c> a +b)
  do This();
  else do That();

If a=b=2, then the P2 Mutant would be the equivalent mutant because they can’t find the applied errors. The Mutation Test has generally high computational costs because many Mutants could be made from main program. Therefore, a need to data test sample that can detect the maximum Mutants is clear. There are a few techniques to reduce the costs, like selective mutation test (L. Bottaci et all, 1999). In this method, the number of Mutants is reduced by declining those with low performances. Mutation Sampling is another technique to reduce the number of Mutants in which a set of them are chosen randomly (M. R. Woodwar et all, 1988).

\[MS(DT) = \frac{\text{whole number of dead mutants} \times 100}{\text{whole mutants-eqivalent mutants}}\]  
(1)
Figure 1, shows the process of the Mutation Test in which all the elements of a mutation system exist Genetic Algorithms are used in Optimizer section (M. R. Woodwar et all, 1988), (Mohsen Fallah Rad et all, 2012b).

**Figure. 1. Mutation testing process.**
III. Genetic Algorithm

The Genetic Algorithm (fig. 2) is the algorithm based on searches according to the natural and biological mechanisms on different environments like learning, (D. E. Goldberg, 1989) searching and optimizing (C. C. Michael et all, 2001).

In software testing, the main search concept in input range is an attempt to reach the optimized goals and the problem is a function like $f(x_1,x_2,…x_m)$ where $(x_1,x_2,…x_m)$ is an adjustable variable in order to get to overall optimization. (D. E. Goldberg, 1989)

These variables are as bit-strings, like chromosomes in nature. In biology, each bit-space is occupied by a gene from a chromosome and each gene consists of some “allele” (D. E. Goldberg, 1989). The Genetic Algorithm, searches the input data range in order to achieve the sample that kills the maximum Mutants.

In order to write a genetic algorithm, we need to define the fitness function that calculates the quality of each chromosome.

The Genetic Algorithm uses three operators: Selection, Reproduction, and Mutation:

1) Selection
This process uses two chromosomes for Generation and Mutation operators (Mutagen) in order to generate new children. There are different methods in order for choosing the parents like the Random Method or related to the Fitness Function (D. E. Goldberg, 1989). In choosing the parents by using the Fitness Function, in contrast with the Random Method, it’s assured that, since the parents are chosen according to their own performance, those with higher fitness value have a better opportunity to be selected.

2) Reproduction
It’s a process where the new children are generated according to the parents’ fitness value and using the Mutation and Combination operators. This operator is a key point in the strength of the Genetic Algorithm. This operator generates the new children according to this idea that they are closer to an optimum answer (D. E. Goldberg, 1989). The crossover of this operator is in the chromosome stage. Before a cross over is applied, chromosomes can be presented into binary form. The simple form of a cross over operator is a single cross over. In this type of cross over, a point is randomly selected between the parents and each of which is divided in two sections of Left and Right. In cross over, the left side for the parents is held constant meanwhile the right sides are substituted.

3) Mutation
The Mutation operator is applied to modify the values of one or several genes. (For example in a bit-string, mutation changes the value from zero to one and vice versa). The mutation, crossover and reproduction have a high strength in searching the input space. Usually in a mutation test, the mutation score is used as the Fitness Function. Using the genetic operators and the fitness function, the genetic algorithm would be as follows:
1) Selection of a primary data set for the primary population

2) Determining the fitness function (the mutation score) for each member

3) Selection of new children and putting them in population as follows:

   A. Choosing the parents with a selection technique
   B. Applying the cross over operator on the couples
   C. Applying the mutation operator on some member

4) Computation of the mutation score for each member

5) If the average mutation score, as the final condition desirable? If yes, the algorithm is terminated, unless it will step backward to number 3.

As the best test data could be accomplished automatically in minimum time, it would optimize the time and computation in operation.

![Diagram of Simple Genetic Algorithm](image-url)

**Figure. 2.** Simple Genetic Algorithm.
IV. Bacteriological Model

This algorithm (fig3) is inspired by the reproduction of the bacteria in nature. In this model, like bacteria, which don’t combine with each other in order to reproduce new children, we don’t have combination anymore and just the Mutation operator is used. In this algorithm, a threshold named BMS (Basic Mutation Score) is used. Chromosomes with mutation score below the BMS should withstand the mutation operator. Chromosomes with scores above BMS will go to the next population without any modification. Lack of a cross over operator in this algorithm increases the rapidity and reduces the time and calculation expenses. Since there are no differences between chromosomes with good or bad mutation, the behavior is similar to random algorithms (B. Baudry et al, 2005), (Mohsen Fallah Rad et all, 2011a), (Mohsen Fallah Rad et all, 2011b), (Mohsen Fallah Rad et all, 2010), (Mohsen Fallah Rad et all, 2012).

Figure. 3. Bacteriological Algorithm.
V.  Recommended Algorithm

This algorithm (Fig. 4 and 5) is a combination of ordinary and bacteriological genetic algorithm. In this algorithm, in each generation, if the chromosome mutation score is not high enough, mutation will continue as far as a proper data is achieved in this mutation and after the parents reach to the desired score, they are combined to produce better children. There are important parameters like mutation rate and crossover rate in genetic algorithms. Mutation rate means what percent of the parents cross over and the mutation rate means what percent of the parents face the mutation. In ordinary genetic methods, these values are usually constant and they can hardly be followed or determined only by a lot of experiments.

In recommended algorithm, these amounts are defined automatically and modified if necessary. It would probably be essential in the primary generations that majority of the chromosomes are periodically facing mutation operation due to having less mutation scores. However, as the mutation score increases in the last generation, this need is decreased. Now there has to be a good crossover action with a good mutation score in order to generate appropriate children. The parameters Nc & Np are the number of children and the amount of population. The amount of Nc in the beginning of the process in Fig. 5 is zero and with each operation between the parents, the value increases.

The BMS value evaluates the modifiability of a pair of chromosomes and using this factor, it could be guaranteed that Chromosomes with high mutation score out of the minimum BMS meet the requirement to go to the next generation.

After each generation, or the new population, the BMS value reduces according to $BMS (i +1) = \lambda \times BMS (i)$. And this operation is equivalent to annealing. Since in primary generations or populations the BMS value is high, a short percent of the parents can operate to reproduce children and many of them have to withstand a mutation. In primary stages, due to high withstanding of the mutation operator, a high diversity is observed in reproduction. As the BMS value decreases in each generation, the number of the parents that have to withstand the mutation operator will be reduce and those that have to cross over will increase. Thus, the recommended algorithm has a good local search in final stages.
Figure 4. Recommended Algorithm

Figure 5. Process of figure 5 from recommended Algorithm
VI. Running

The mutation test system is written in JAVA environment and SQL 2005 is used in order to record and restore the data.

<table>
<thead>
<tr>
<th>Method</th>
<th>Number of Produced Generations</th>
<th>The average scores of mutations of the final data (%)</th>
<th>The equivalent mutants found</th>
<th>The average number of killed data by the final data</th>
<th>The average number of un-killed data by the final data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recommended algorithm</td>
<td>18</td>
<td>98.10</td>
<td>45</td>
<td>71</td>
<td>166</td>
</tr>
<tr>
<td>BMS=50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recommended algorithm</td>
<td>8</td>
<td>92.30</td>
<td>45</td>
<td>77</td>
<td>160</td>
</tr>
<tr>
<td>BMS=65</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Normal GA</td>
<td>17</td>
<td>90.66</td>
<td>45</td>
<td>77</td>
<td>160</td>
</tr>
<tr>
<td>Bacteriological</td>
<td>16</td>
<td>87.50</td>
<td>47</td>
<td>81</td>
<td>154</td>
</tr>
<tr>
<td>BMS=65</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A. The application in use
The triangle detection program (TriType) which is the standard application and is used in most of different software tests thanks to having mutative and conditional skips. The input includes three numbers as the sides of the triangle and the output is either 1 as a scalene, 2 as an isosceles, 3 as an equilateral or 4 as it’s impossible to form a triangle using the given numbers. 282 mutants are generated using this application.

B. Choosing the parents
In order for the parents to be chosen, a roulette wheel is used. Chromosomes are placed on the roulette wheel and according to their fitness values; they occupy somewhere especial on the wheel. Then the wheel is rotated and after stagnation, a chromosome is picked up. It’s clear that chromosomes with higher fitness value are more likely to be selected.

10 bits are considered for each side of the triangle. In this simulation, as each chromosome as a test data equates three sides of a triangle, it occupies a space of 30 bits. The lengths of each population are constant and equate 6 chromosomes.

C. Simulation Results
Observing table 1 leads to the following results:

1) The number of reproduced generations in the recommended algorithm is less in comparison to other methods, because the input space in primary generations is sufficiently verified by consecutive mutations.
2) Although the number of killed mutants by the data of the final generation of Bacteriological algorithms is higher in comparison to other methods, the average mutation score for this data is less than them.
3) The numbers of equivalent mutants in bacteriological method are more than other methods by two numbers. It means there are two mutants that are killed in other methods but they’re not found here.
4) The results prove that the time and calculation cost of the recommended algorithm is less than other techniques.

VII. Conclusion
In this article, we’ve managed to reduce the expenses of test data developments and replacement in a mutation test using the modifications in the Genetic Algorithm and crossover with bacteriological mode. This happens thanks to the dynamic modification of mutation rate and crossover rate by the algorithm itself referring to internal conditions which is in contrast with the previous techniques.
References

i. B. Baudry, F. Fleurey, J. Marc, Y. Le Traon, (2005), "From genetic to bacteriological algorithms for mutation-based testing", Software testing, Verification and Reliability, Published online 4 January 2005 in Wiley Inter Science, pages 73–96.


iii. Mohsen Fallah Rad, Mohadeshe Moosavi, (2011)" Introduction of Mutation Testing and Its Cost Optimization Methods by Focusing on Improving the Test Data Set", ICCTD.


v. Mohsen Fallah Rad, Mohadeshe Moosavi (2011) ," A review on using of Quantum Calculation Techniques in optimization of the data system of mutation test and its comparison with Normal Genetic Algorithm and Bacteriological", ICCTD.


ix. Mohsen Fallah Rad, Sajad Bahrekazemi, Mazdak Behzad, mohamad Reza


xi. Mohsen Fallah Rad, Mohamad Fallah Rad, Sajad Bahrekazemi (2012), Introduction of Mutation Test and Acquaintance with its Advantages and disadvantages by Focus on Reducation of its Costs with Using of Evolutionary Algorithms”, the 2nd Lahijan’s National Conference on Software Engineering, Lahijan Branch, Islamic Azad University, Lahijan, Iran, Lnce.