International Journal of
Soft Computing and Software Engineering (JSCSE)

Vol.4,No.1, 2014
Published online: Jan 25, 2014

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v4.n1.1

# Applying Genetic Evolutionary, Bacteriological and Quantum Evolutionary Algorithm for Improving Performance Optimization Segment of Test Data Sets in Mutation Testing Method

Mohsen Fallah Rad*[1], Sajad Bahrekazemi[2]
[1,] Department of Software Engineering, College of Engineering, Lahijan Branch, Islamic Azad University, Lahijan, Iran.
[2,] 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.
Email: [1]mfalahrad@gmail.com, [2]Sbahrekazemi@ymail.com

**Abstract.** ***Due to computer progress, computer systems became bigger and their function area expanded too. So, software testing, as a part of software engineering, has gained great importance. The goal of software testing is improving software quality and being sure about the accuracy of the final product; moreover the programmers test it for evaluating software accuracy. There are a variety of methods for software testing, among which, the mutation testing is the most famous one. In this method, high range mutants are made from the original program, and then attempts are made to discover mutants by the help of testing data collections. Whenever necessary, testing data can be improved or software deficiency can be found in the process of making and discovering mutant. This is done through algorithm of genetic evolutionary, bacteriological, particle swarm optimization, and evolutionary quantum, which have a high quality for research and can be done automatically. In these methods, test data can be improved by using the properties of the above evolutionary algorithms and without any human intervention in optimization part of the test data of mutation testing system, which consequently leads to a huge reduction in mutation testing costs. In these methods, test data can be improved by using the properties of the above evolutionary algorithms and without any human intervention in optimization part of the test data of mutation testing system, which consequently leads to a huge reduction in mutation testing costs. In this article, four methods of algorithm of genetic evolutionary, bacteriological, particle swarm optimization, and evolutionary quantum have been studied for improving testing data in mutation.***

**Keywords**: *software testing, mutation testing, genetic algorithm, bacteriological algorithm, particle swarm optimization, evolutionary quantum*

* Corresponding Author:
Mohsen Fallah Rad
Department of Software Engineering, College of Engineering, Lahijan Branch, Islamic Azad University, Lahijan, Iran
Email: mfalahrad@gmail.com Tel: +98-141- 2229081 (234)

## 1. Introduction

Software plays an important role in human's life. So there's a great pressure on software engineer for making different and user-friendly software, because low quality software leads to wasting time and probably lack of people eagerness. Therefore, software testing is a process of discovering software deficiencies to improve its quality [6]. Software testing is a costly process in software companies and about 30% to 40% of developing budget is usually allocated to finding errors before introducing the

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v4.n1.1

software to the market [6]. Software testing, though costly, is so important and necessary and it is also important as a part of software engineering. There are many testing methods that their goals are improving software quality and being sure of the accuracy of the final product. In this article, mutation testing has been used for software accuracy. Mutation techniques are generally used for improving software quality. Needless to say, this method includes high costs such as testing data replacement, several running of the testing data on mutants, comparing running result of data samples on the original program and mutants, and their analysis. Today's experts are trying to reduce the mentioned costs.

Some software can conceal failures in the process of testing, so by the help of mutation testing, these failures can be found. Mutation testing leads to discover potential and logical failures in the software, by improving the quality of testing data samples and also smart discovering of mutant[10],[11].

One of the most expensive parts of the mutation test is recognizing the inadequacy of the test data which as a result, one must improve test data and run mutation testing system again. Every method that can accelerate achieving the desired data for a test has a significant impact on reducing the cost of mutation testing. In this article, the effect of using evolutionary algorithm such as genetic, bacteriological, and quantum in reducing the costs of this method has been shown.

## 2. Mutation Testing

### 2.1. Mutation testing definition

Mutation testing is an attempt to solve the problems related to calculation of testing data set accuracy and it is a reverse solution for improving testing data or software quality. This method has been created by “Holmet” and “Demilo” and its goal is making a testing data collection which can discover maximum mutants from the original program. [12].

Since mutation testing analyzes each code line of the program, different route of program, and circulation of testing data, it is a subset of White Box testing. White Box testing uses the information related to system internal mood for conducting examiner and testing [13]. Mutants are simply made by mutagens. Mutagens can make simple errors in the context of the original program by changing mathematical and logical operator, changing constants, changing variable, and….

For an instance, some mutagens which have been used in “Mothra” system are as follows [14]:

| | |
|---|---|
| AAR | Array reference for Array reference Replacement |
| ABS | Absolute value inSertion |
| ACR | Array reference for Constant Replacement |
| AOR | Arithmetic Operator Replacement |
| ASR | Array reference for Scalar variable Replacement |
| CAR | Constant for Array reference Replacement |
| CNR | Comparable array Name Replacement |
| CRP | Constant RePlacement |
| CSR | Constant for Scalar variable Replacement |
| DER | DO statement End Replacement |
| DSA | DATA Statement Alterations |
| GLR | GOTO Label Replacement |
| LCR | Logical Connector Replacement |
| ROR | Relational Operator Replacement |
| RSR | RETURN Statement Replacement |
| SAN | Statement ANalysis |
| SAR | Scalar variable for Array reference Replacement |

For example LCR (Logical Connector Replacement) operator can replace every event which is a part of logical operator.

Nowadays, mutagens are used for objective oriented language.

| | |
|---|---|
| EHF | Exception Handling Fault |
| AOR | Arithmetic Operator Replacement |
| LOR | Logical Operator Replacement |
| ROR | Relational Operator Replacement |
| NOR | No Operation Replacement |
| VCP | Variable and Constant Perturbation |
| MCR | Methods Call Replacement |
| RFI | Referencing Fault Insertion |

After gathering testing data and mutants, an attempt is made for discovering mutant by testing data sets. In other words, if the answers of running testing data on the original program and a mutant are not the same, the mutant would be considered as a killed mutant; but if the answers are the same, then they have one of these conditions:
A) It is an equivalent mutant which means there is a similar answer to the original program for every testing data sample. The production of equivalent mutant must be banned through a correct and precise programming; because the program must be sensitive to change and insert errors in the context code.

B) Mutant is not really equivalent but it has the same answer as the original program due to the deficiency of testing data sample. This fault must be removed by improving and reinforcing testing data sets.

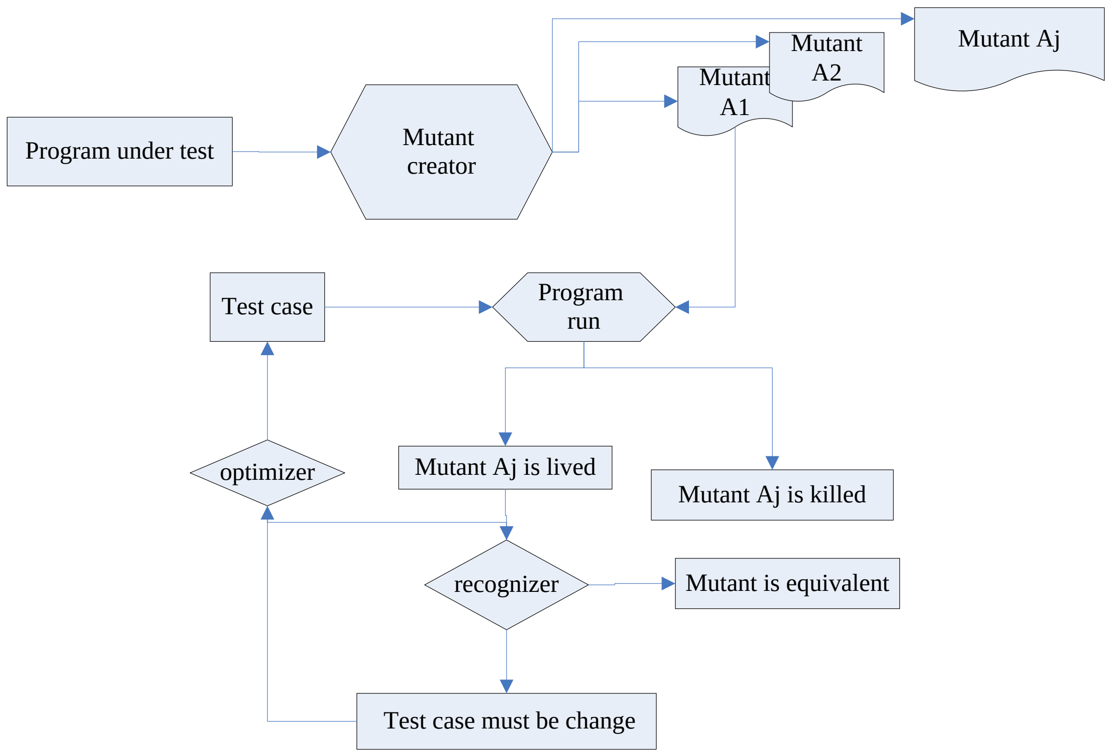

Figure 1. Mutation testing process

Mutation testing is based on two principles:
1. Skillful programmers, because they have the least logical errors. In other word, mutation testing can discover illogical and small errors which are hidden from the examiner’s eyes and it cannot

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v4.n1.1

show the logical errors. So, due to high cost of testing, using this method is just recommended when necessary [14].

2. Couple effect: 'Offut' proved that if testing data sets can discover the maximum errors resulted from the mutagens, and then they can discover more complicated errors with a high percentage. Mutagens use mutation score for calculation of the quality of testing data sets [14].

$$MS(DT) = \frac{whole\ \ number\ \ of\ \ dead\ \ mutants}{whole\ \ mutants - equivalent\ \ mutants} \times 100$$

### 2.2. Example:

If source code is like below and then it is easy to make mutants M1, M2, M3, M4 from it by mutagens

P: Max = a;
If (max < b)
Max = b
If (max < c)
Max = c

Then mutants are M1, M2, M3 and M4:

**M1: Max = b;**
If (max < b)
Max = b
If (max < c)
Max = c

**M2**: Max = a;
***If (max > b)***
Max = b
If (max < c)
Max = c

**M3**: Max = a;
If (max < b)
***Max = a***
If (max < c)
Max = c

**M4**: Max = a;
If (max < b)
Max = b
***If (max < a)***
Max = c

## 3. Mutation testing costs

Mutation testing is basically costly, because testing data sample must be run on a large amount of mutants and then results must be compared to the original program. If the number of killed mutant is not satisfying, testing data sets must be improved, and again the whole mentioned process repeats for one more time. Further changes and running need high cost. In addition to calculation and running cost, those costs related to discovering equivalent mutant and destroying live mutants (due to the weakness of testing data sets) make this method more expensive.

## 4. Solution for reducing mutation testing costs

Beside basically mutation testing costs, both in the selection step of testing data and running step, other costs include code changing in the case of not gaining desired answer and re-run costs of the testing. Researchers have tried and invented variety of methods for reducing costs.

### 4.1. Solution for reducing calculation and operation costs:

It is mostly focused on the cost reduction of producing testing data sample and is searching solution for reducing numbers of mutants and faster running.

#### 4.1.1. Do-fewer method

It insists on the reduction of the mutants. Selective mutation testing is one of the most famous methods based on this idea [14].

In this method, mutagen operations are divided into three main branches such as replacement operation operator, phrase correction operator, and proposition correction operator, and mutation testing can be selected from each branch or a combination of them. The results have shown that by

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v4.n1.1

using this method, the score of testing data collection will be more than 95%. In the other words, by reducing mutagens and due to the condition, it leads to the reduction of running costs with non-selective mutagens score [14].

#### 4.1.2. Do-smarter methods

It focuses on the smart usage of the operators and internal mood of the running program. One of the best methods here is called Weak Mutation Testing.

“Leonarelo” is a revised version of “Mothra” which has been built on weak mutation testing principles [14]. In this method, testing is more based on analytical natural form than running. In other words, instead of a real running of mutation, it decides analytically about their killing. This analysis can be done about control flow content, program counter stables, results between stored blocks in the program, variable space, basic stable block, instruction, and etc.

#### 4.1.3. Do-faster methods

It insists on the faster running of the mutants. One of their most famous one is Schema Base Mutant [12]. The base of this method is creating special parameters as meta-mutant, which is made in compiler level, and so is much faster than other methods. As an example, it includes meta-mutant ARQ, and 6 mathematical operator * ،/ ،+ ،- ،Mod ،Sqrt.

#### 4.1.4. Optimistic Mutation system

One of the most expensive parts of mutation testing is finding equivalent mutants. It has been mentioned in this method that if a system can kill 95% to 99% of the mutants and it is acceptable for the examiner, then the other mutant, equivalent or alive, can be ignored, so the costs of discovering equivalent mutant is eliminated [12].

#### 4.1.5. A method based on the threshold level

A threshold level is determined for a system and for testing data which equals to minimum average of the mutation score that they must gain. If they don’t reach to threshold level, those data sets that aren’t able to kill any mutant will be eliminated, and this process continues until reaching to the favorite threshold level.

#### 4.1.6. Automatic analysis system

In this system, human interfere significantly decreases which leads to a decrease in the costs, and Schema producer automatically does meta mutant, and the launcher sample runs testing data on the meta- mutant, and the whole process is done by the machine.

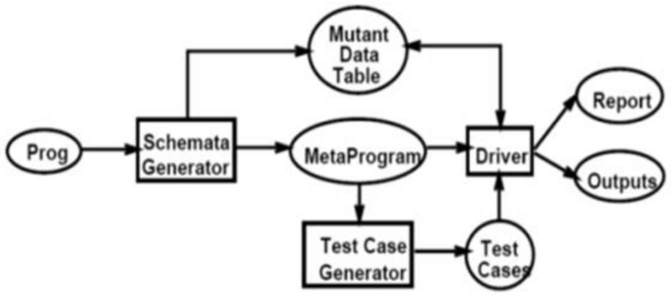

Figure 2. Architecture of the mutation analysis system

### 4.2 Reducing the costs by the help of proper testing data

When the testing data sets cannot kill convincing amount of the mutants, it must improve them. Improving testing data of effect is the most expensive part of mutation testing. In figure

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v4.n1.1

1, an optimizer does it, and due to the importance of this part, evolutionary algorithms are mostly used.

### 4.2.1 Genetic algorithm

It was created by “John Holland” and developed by him and his students. The result of this effort was a book named “Adaptation of natural and smart system” which was published in 1975. In nature, the variety of creatures results from chromosomes. In reproduction process, first intercourse or cross over is done and gens are transmitted to new chromosome through parents in different ways; then new chromosomes must be altered and get mutation. Mutation operator means DNA elements of a particle can be altered and maybe, by this mutation, chromosomes stay in a better condition than their parents, and get free from local peaks. [7][8].

Genetic algorithm is as below:

1. Selection of random primary population of chromosome. (Chromosomes are the same as testing data samples).
2. Calculation of the suitability or efficiency level of each chromosome (suitability function is the same as mutation score function).
3. Making a new population as below:

- Two chromosomes are selected based on a roulette wheel. On this basis, two chromosomes which have more mutation scores have more chance to be selected.
- Some percentages of the chromosomes combine by cross operator. In this operation a point is selected as a combination point and parents are replaced by each other in the point.

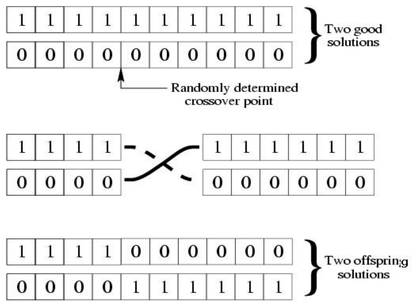

Figure 3. One point cross-over

- Children gain mutation with a probability. A point is randomly selected on the chromosome, and then its content is reversed.

4. New children replace parents.
5. The suitability amount of the new children is calculated and if the average is not suitable or the final condition is not met, we come back to step 3, or the algorithm finishes.

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v4.n1.1

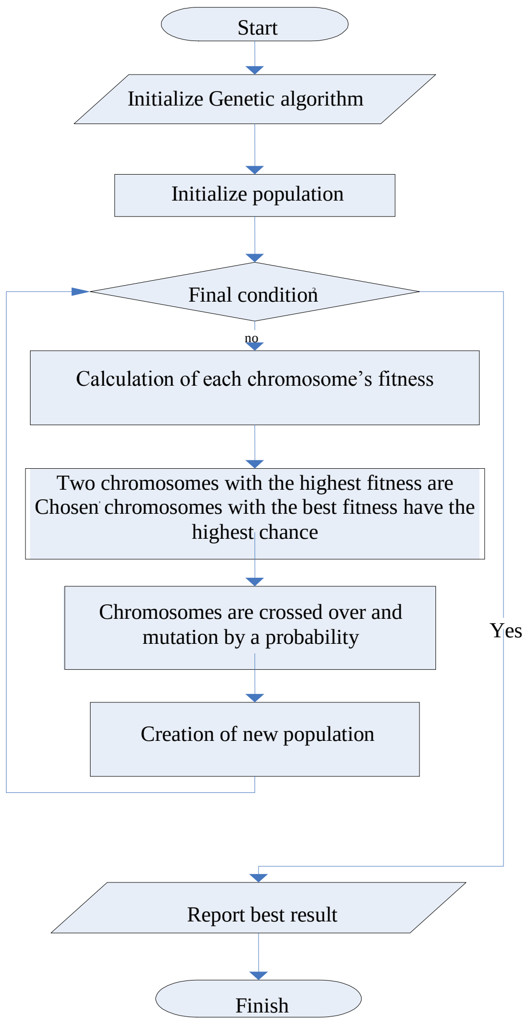
Start
Initialize Genetic algorithm
Initialize population
Final condition
no
Calculation of each chromosome's fitness
Two chromosomes with the highest fitness are Chosen chromosomes with the best fitness have the highest chance
Chromosomes are crossed over and mutation by a probability
Creation of new population
Yes
Report best result
Finish

Figure 4. Genetic Algorithm [3], [4]

### 4.2.2. Bacteriological algorithm [7]

This algorithm is inspired by reproduction of bacteria in the nature. In nature, bacteria does not combine, instead parents only get mutation for producing new child. A BMS (basic Mutation Score) is defined, and chromosomes must bear operation so that their scores go higher than BMS and so transfer to the next generation. Unlike genetic algorithm, there is no combination in this algorithm, therefore its related costs are eliminated too [7], [8].

Bacteriological algorithm is as follows:

1. Selection of primary set of chromosomes (chromosome is the same as testing data).
2. Calculation of suitability of each chromosome (based on mutation score function)
3. Preserving the best chromosome
4. Reproducing by selecting the best chromosome
5. Mutation

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v4.n1.1

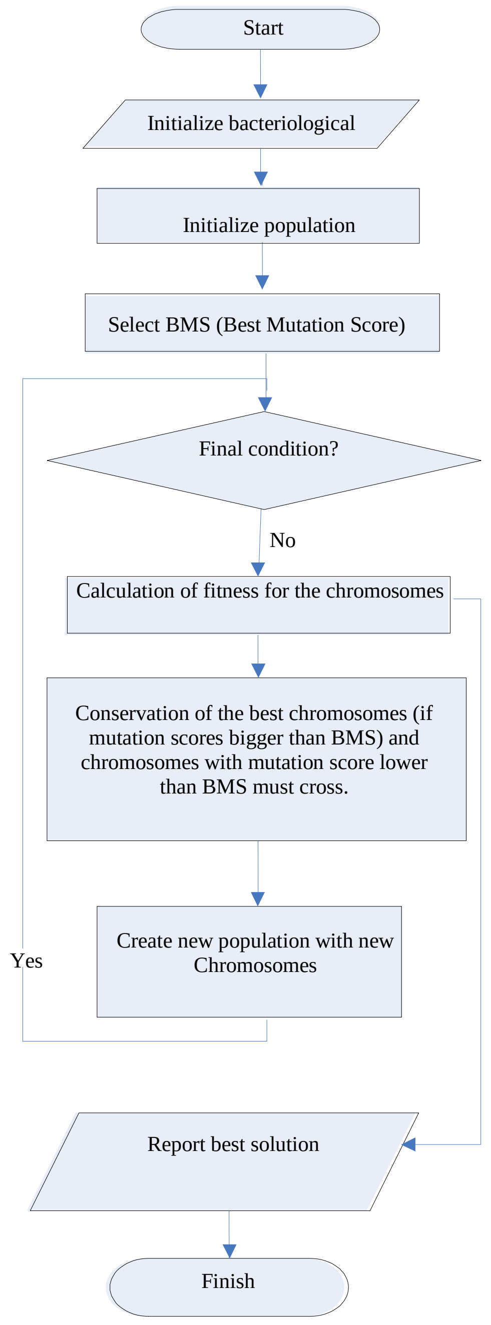
Start
Initialize bacteriological
Initialize population
Select BMS (Best Mutation Score)
Final condition?
No
Calculation of fitness for the chromosomes
Conservation of the best chromosomes (if mutation scores bigger than BMS) and chromosomes with mutation score lower than BMS must cross.
Create new population with new Chromosomes
Yes
Report best solution
Finish

Figure 5. Bacteriological algorithm [3], [8]

Another difference with genetic algorithm is elitism in which elite chromosomes (chromosomes with high mutation score BMS) enters to new generation without change.

### 4.2.3. Particle Swarm optimization Algorithm

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v4.n1.1

PSO algorithm is an optimized algorithm based on the social behavior of birds or fish. This was first introduced by “Kennedy” and “Eberhart” in 1995, and was inspired by social behavior of birds and fish when they are searching food [5], [15]. In every stage of movement, population of each particle gets updated by two best amounts. The first answer is the best answer from suitability point of view which was gained for each particle, separately and is named “Lbest”. Another amount is the best amount which was gained by all particles in the population up to now. This best amount is general and called “Gbest”. Lbest and Gbest amounts are selected based on an evaluating function. This function is variable in different problem. After finding two amounts of Xid and Vid, which respectively indicates special condition and speed related to dimension “d” from particle “I”, new speed and space are updated.
This process is done by below equation:

$$VI[t+1] = wVi[t] + c1 \times rand()\,(Lbest[t] - Xi[t]) + c2 \times rand()\,(Gbest[t] - Xi[t]) \quad (1)$$

$$Xi[t+1] = Xi[t] + VI[t+1] \quad (2)$$

In above equation “w” is the inertia weigh, c1 and c2 are learning factors which are also called velocity factors, and rand() is a random number between 0 and 1. The amount of these parameters is different in variant problem solving. The amount of these parameters is also so effective in convergence of the problem. A main problem of this algorithm is losing variety after hasty convergence and sticking in the local optimum.
PSO and GA similarity is that both of them start with a random population matrix. In PSO each row of the matrix is a bird and is called particle, like a chromosome in GA.
This algorithm is used because of an almost high convergence speed. PSO algorithm is as follows:
1. Primer amount giving of the population
2. Process below continues until having satisfactory result.
1. Calculation of suitability amount of each particle (based on the mutation score).
2. Correction of the best particles in the population.
3. Selection of the best particles.
4. Calculation of particles speed.
5. Updating condition of each particle.

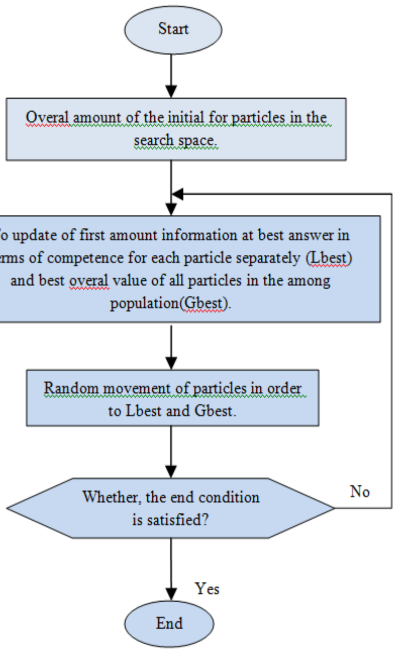
Start
Overal amount of the initial for particles in the search space.
To update of first amount information at best answer in terms of competence for each particle separately (Lbest) and best overal value of all particles in the among population(Gbest).
Random movement of particles in order to Lbest and Gbest.
Whether, the end condition is satisfied?
No
Yes
End

Figure 6. PSO flowchart algorithms

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v4.n1.1

### 4.2.4. Quantum evaluation algorithm [1], [2], [9]

It is formed based on quantum calculation and computers. Its function is the same as bacteriological algorithm but with a better possible structure which leads to a better convergence percentage in the answer population. In each quantum system, a two dimensional vector space is modeled, which is called Q bit. Q bit in an assessable mood Selected arbitrary and tagged <0> or <1>. Classical bits are 0 and 1, while the final amount of Q bit is a synthesis of a<0>+b<1> function in which a and b are complex numbers that finally and by a measuring function, their final amount become 0 or 1. A rule which explains the method of the result extraction from quantum state first was introduced by "Marx Born" and now it is known as 'Born' rule.

The algorithm stages are as follows:

1. In the first step, the probability of all Q bit chromosomes is considered equal. It means the probability of observing zero and one in all Q bits are considered ass equal.
2. New stage chromosomes by the observe function $U(\Delta\theta)$, circulation function $\theta$ under a very small angle, is as follows:

$$x_i = \begin{cases} 0 & U(0,1) \prec \alpha_i^2 \\ 1 & otherwise \end{cases}$$

3. Mutation score of each chromosome is calculated.
4. Like the bacteriological algorithm, the best chromosomes transfer to the new stage, and those chromosomes with mutation score under condition of passing the stage (BMS), must be affected by some quantum operators to reach a score upper than BMS.
5. After formation of the new generation, if the mutation score average of all chromosomes is higher than the final condition, then the algorithm finishes. In other case, we must calculate the factors of new extension of state function based on a circular function and go to stage 4.

## 5. Simulation result

Simulation is in MATLAB and Java programming and used for storage and retrieving information from SQL. The used program is Tritype diagnosis which is used in almost all testing as a standard program, due to its conditional jump. Its input is three numbers which stands for three sides of the triangle. Its output is 1, which means three unequal sides, 2, which mean Isosceles, 3, that means equilateral, and 4, that means these three sides cannot make a triangle. Mutant can easily be formed. Each chromosome indicates a side of a triangle in which 10 bit is allocated to each side. In other words, the length of each chromosome is 30 bit and mutation score of each chromosome is calculated.

Roulette wheel is used for selecting parents in the genetic algorithm. While in the bacteriological algorithm, PSO, and quantum evolution, parents who have conditions upper than BMS are automatically transferred to the next stage.

In Table 1, mutation score average of genetic algorithm, bacteriological algorithm, and PSO is studied in 10 generation in which genetic algorithm in fifth, sixth, and eighth generation shows a better mutation score. Bacteriological algorithm has a better mutation score in first, second, third, fourth, and seventh generation compared to other algorithm, but PSO algorithm shows a better mutation score in the ninth and tenth generations and increases progressively.

Based on the table 2, the average of the killed data, also its mutation score in PSO algorithm is more than the average of the other algorithms. Moreover, the number of equivalent mutant is lower than other algorithm which is an indication of more mutant killing in this method.

In the genetic algorithm, the average of the killed data number is lower than other algorithms, but its mutation score is more than bacteriological algorithm. Moreover, the number of its equivalent mutant shows one number difference compared to bacteriological algorithm. This difference indicates that there is one mutant which has been killed by the genetic algorithm, but is considered equivalent in bacteriological algorithm by mistake.

Bacteriological algorithm has a lower mutation score average than other algorithms. The numbers of equivalent mutants of this algorithm are also more than other algorithm which shows lower killing mutant in this method. Based on the above convergence graph in the genetic algorithm, due to

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v4.n1.1

simultaneous running of combination and mutation operator on chromosomes of each generation, there is a possibility of up and down. Moreover, based on the table 1, after the seventh generation, chromosomes find proper pattern of getting optimized answer and gradually incline to these answers. The manner of bacteriological algorithm, PSO, and quantum evolution is the same, and similar to the random algorithm. Therefore, some ups and downs can be seen in them.

***Table 1.*** : Mutation score average in every generation of genetic algorithm, bacteriological algorithm, and PSO

| **PSO Model** | **Bacteriological Model** | **Genetic Model** | |
|---|---|---|---|
| Average of mutation score based on percentage | | | Input data |
| 23/55 | 24/66 | 23/32 | Primary data |
| 34/33 | 47/83 | 38 | First generation |
| 42/53 | 50/66 | 44 | Second generation |
| 28/87 | 44 | 33/21 | Third generation |
| 44/16 | 55/16 | 45/22 | Fourth generation |
| 48/12 | 50/33 | 54 | Fifth generation |
| 42/22 | 44/83 | 50/3 | Sixth generation |
| 38/20 | 48/66 | 35/5 | Seventh generation |
| 48/21 | 45/83 | 65/9 | Eight generation |
| 58/43 | 46/83 | 55 | Ninth generation |
| 62 | 58/33 | 60/2 | Tenth generation |

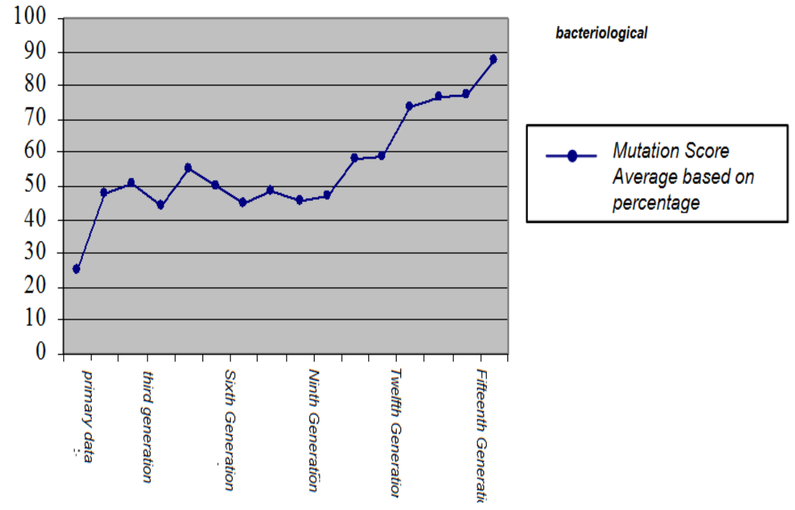

Figure 7. Convergence graph of the genetic algorithm

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v4.n1.1

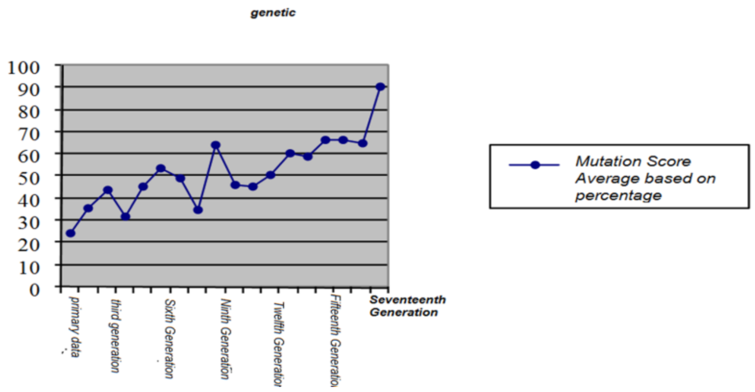

Figure 8. Convergence graph of the bacteriological algorithm

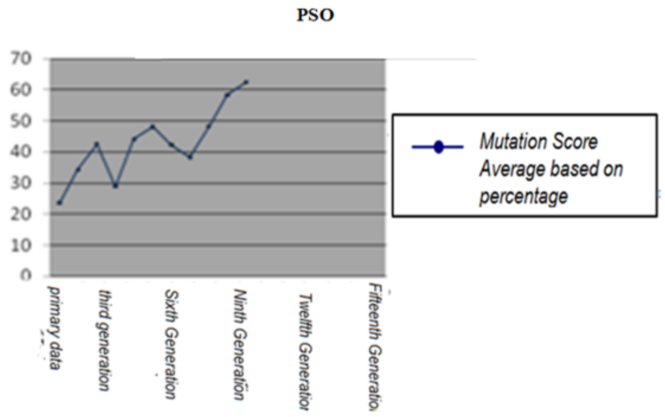

Figure 9. Convergence graph of the PSO algorithm

In the genetic evolution which is inspired by real gens, change and evolution happen slowly and in many years. Maybe this is because the genetic algorithm needs more repetition for finding the optimized answer. On the other hand, PSO algorithm operates faster due to inspiration from birds that operate so faster for their food and eating. It is also faster in problem solving then the genetic algorithm.

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v4.n1.1

***Table 2.*** Comparison of three algorithms

| **Average of not killed data by the final data** | **Average of killed data by the final data** | **Found equivalent mutant** | **Mutation score average of final data** | **Number of produced generation** | **method** | |
|---|---|---|---|---|---|---|
| 79 | 35 | 26 | %60/2 | 10 | **Genetic Algorithm** | |
| 76 | 36 | 27 | 58/33 % | 10 | **Bacteriological Algorithm** | |
| 74 | 38 | 25 | % 62 | 10 | **PSO Algorithm** | |

## 6. Conclusion

Mutation testing is one of the most important methods of software testing. This method has many advantages compared to the other methods of software testing. For example, latent errors, which cannot be observed easily, will automatically be identified. Moreover, this method has basically too much calculation, operation, and timing costs, and so must be used only in economically justified conditions. Many researchers are trying to reduce the cost of this method and this reduction can be done in two dimensions. First, reducing the cost of running and testing in which Do-smarter, Do-fewer, and Do-faster can be mentioned as the most famous methods. Two, attempts to faster improve of testing data, such as genetic evolution algorithm, bacteriological algorithm, PSO algorithm, quantum evolution, and etc.

## 7. Future plans

In this article, the stages of the usage of these algorithms are explained efficiently by the help of a practical example. The mutation testing system and algorithms are defined for testing small programs or units. For testing bigger programs or systems, more focus should be made, that can be taken into account for future studies.

Despite effective results of using evolutionary algorithm in cost reduction of mutation test, their usage in very large programs, object oriented, or small or medium programs which own many leaps, has many limitations, because evolutionary algorithms lead us to certain optimized solutions which are convergent, while we need to certain optimized solutions which are not convergent and can search all paths of the program and at the same time, use the properties of the evolutionary algorithm. Convergence of the evolutionary algorithm, and also its automaticity and practicality are big challenges which can be studied in future researches.

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v4.n1.1

## References

[1] N. David Mermin, "quantum computer science", Cambridge University Press, 2007.
[2] A. khorsand, M. Akbarzadeh, "quantum gate optimization in a meta-level genetic quantum algorithm", 2006.
[3] Mohsen Fallah Rad, Farshad Akbari, Ahmad Javan Bakht," Implementation of Common Genetic and Bacteriological Algorithms in Optimizing Testing Data in Mutation Testing", cise.2010.
[4] Mohsen Fallah Rad, Mehdi dehghan, ahmad javan bakht,," optimization data sets in mutation testing with perfect genetic algorithm",isfs2008
[5] J. Kennedy, R. Eberhart, "Particle Swarm Optimization", IEEE International Conference on Neural Networks, Perth, Australia, page 1942-1948, (1995).
[6] k. Beck, E .Gamma, " Test-infected: Programmers love writing tests", Java Report1998, pages 37–50
[7] B. Baudry, F. Fleurey, J. Marc, Y. Le Traon, "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.
[8] Mohsen Fallah Rad, Mohadeseh Moosavi," Introduction of Mutation Testing and Its Cost Optimization Methods by Focusing on Improving the Test Data Set", ICCTD 2011.
[9] Mohsen Fallah Rad, Mohadeseh Moosavi," 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 2011.
[10] W. M. Craft, "Detecting equivalent mutants using compiler optimization techniques", Master's thesis, Department of Computer Science, Clemson University, Clemson SC, 1989
[11] M. R. Woodward, K. Halewood. "From weak to strong, dead or alive?, An analysis of some mutation testing issues", In Proceedings of the Second Workshop on Verification, and Analysis, Ban Alberta, IEEE Computer Society Software Testing, Press, July 1988, pages152-158.
[12] A. j. Offutt, "Practical system for mutation testing: help for the common programmer", National science foundation under grand ccr-93=11967, 1993.
[13] Myers, G.J., "The Art of Software Testing", John Wiley & Sons, New York, 1979.
[14] R. A. DeMillo, E. H. Spafford, "The Mothra software testing environment ",Proceedings of the 11th NASA Software Engineering Laboratory Workshop, Goddard Space Center, December 1986.
[15] V.K. Patel, R.V. Rao, “Design optimization of shell- andtube heat exchanger using particle swarm optimization technique,” Applied Thermal Engineering, Vol.30(11-12),August 2010, pp.1417-1425.