Investigation of Improving Test Data in Mutation Testing by Optimization Methods

Mohsen Falah Rad and Mohadeseh Moosavi

***Abstract*—One of the major expenses about creating and developing a software, is the testing expenses; of which, one of the most important methods is the Mutation Testing. In this method, several versions of the original program including various errors are created and using the appropriate test samples, it's tried to discover the mistaken versions. The more proper the program and samples, the more number of mistaken versions will be discovered. In order to simulate this method, we need to design a system that executes the test data samples on many mistaken version and then through comparing these results with the main program, discover the versions with error. This method naturally has high calculative and operational costs. The cost includes, repetitive execution of each test data, creating versions with errors and modification of the test samples and mistaken versions if necessary. There's a general method to reduce the costs. The first category includes the methods which rely on performance expense reduction or reduction of the number of mistaken version. The second rely on fast and cheap data samples of the test that leads to reduction of comparisons between the output of the main program and mistaken versions. In this article, we're to make a comparison between the methods and introduce the system of mutation tests in order to improve the test data according to evolutionary algorithms.**

***Index Terms*—Mutation testing, genetics algorithm, bacteriological algorithm, mutation testing's cost.**

I. Introduction

Software engineering test today has been an important component of software engineering. It has been of particular importance but also very costly, including about 50% of the expenses. In general, techniques of software testing are divided into two groups: static and dynamic [1], [2]. In static testing, source program observer analyzes the program in parts and by observing and following the inputs. In dynamic testing a series of testing data is performed on the program and then the output is observed and analyzed.

Dynamic testing is divided into two groups: Black box and White box. White box testing analyzes the program structure, the code of the program and the current of data in the program [3]. In white box testing, the information concerning internal system moods is used to lead the test and tester. In black box testing, it's not necessary to have some information concerning the internal structure of the system. In this technique, samples of testing data is created based on system features and it is just needed to have some information concerning what inputs are needed and what outputs are expected[3].

Manuscript received November 9, 2011; revised December 11, 2011.

The authors are with form Lahijan Branch, Islamic Azad University, Lahijan, Iran

The aim of this software testing is to develop the software quality and ensure the achievement of obtaining a fine output [4], [5], [6].

One of the methods to develop the data test quality is to introduce smart errors in a test that is called Mutation Analysis [7], [8], this method classified into the white box testing. In this method, the quality of the data test is proportional to finding error contained programs. These programs are called Mutants. The value of the test sample is proportional with the quality of the data test (the discovery power or the mutant detection).

A. The Description of Mutation Testing

This technique was first used by "Holmet" and "Demilo" and the goal is to try to establish the test samples that can recognize the maximum faulty versions of the main program [4], [6].

We can simply use the mutation operators to create many error contained programs from the main programs. These mutants are created by modifying the mathematical, logical, and changing in constant values, changes Variable and Operators from the main program and etc. After that, it's investigated whether this set of data can find the difference between this faulty versions and the program output. In this form, it could be realized that the test data set are qualified to test the program and also this program is in a fine form of examinablity, logic, error detection. According to the modification in the original program, the data test has been able to understand these modifications. Otherwise it could be concluded that the data test aren't selected appropriately. Thus, we must use the optimization techniques to modify the data test or the program lack the right logic, which in this case, the program logic, the codes and the ambiguities have to be corrected with more attention[4], [6].

By getting an output from the main program and the concluded results from the faulty versions, and comparing them we could discover error contained programs. In this way, two goals of evaluating the data test quality and error detection could be achieved simultaneously. The Mutation-based techniques do their effort to improve the software quality but majority of these techniques are expensive and have high calculation and time cost. Because we have to have many faulty versions of the main program and compare the results of these versions and the main program. These procedures can continue to the desired results over and over therefore it could be concluded that the mutation test is a costly test that will be used just when a high accuracy is required and the programmer needs a program completely out of errors [9].

The mutated programs are made using the mutation

operators which make a small modification in the code or the main program operators. As an example, some mutation operators of the "Mothra" mutation system, which is written based on FORTRAN are as follows [10]:

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, With AOR operator, changes each mathematical operator with another one. Today, the mutation operators are made for objective languages [9].

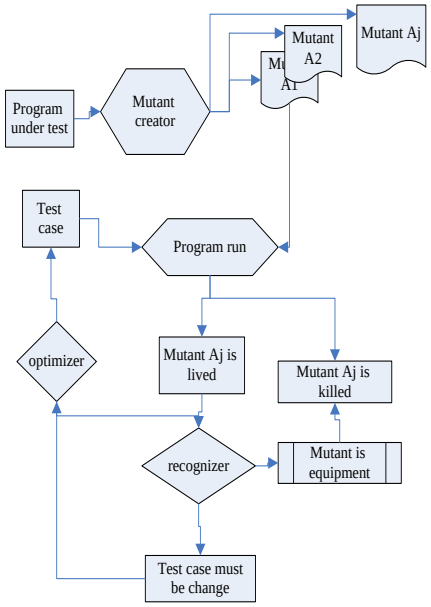

Fig. 1 . The Quantitative process of creation and developing the test's data set in mutation testing [10].

Mutation Testing is based on two principles: 1) a skilled programmer: The program written by such a programmer has the minimum logical errors. Therefore, the slight errors could be found by Mutation Tests [11]. 2) The coupling effect "Offut" has proved that if a data test can identify all the mutants made by a simple change, it can find the more complicated errors to a desirable part [11].

In the Mutation Test, the Mutation Score is used in order to calculate the quality of the data test set in a formula like this:

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

Member of the test's data set and the performance could be measured for each of the members [12].

B. Examples

If $r:1;$

$for\ i = 2\ to\ 3\ do$

$if\ \ a[i] > a[r]\ \ than\ \ r := i,$

These mutants can easily be made:

$m1)\quad r:1;$

$for\ i = 1\ to\ 3 do$

$if\ \ a[i] > a[r]\ \ than\ \ r := i,$

$m2)\quad r:1;$

$for\ i = 2\ to\ 3 do$

$if\ \ i \geq a[r]\ \ than\ \ r := i,$

$m3)\quad r:1;$

$for\ i = 2\ to\ 3 do$

$if\ \ a[i] >= a[r]\ \ than\ \ r := i,$

$m4)\quad r:1;$

$for\ i = 2\ to\ 3 do$

$if\ \ a[r] > a[r]\ \ than\ \ r := i,$

II. Mutation Testing Costs

Mutation testing by itself has high calculation and time expenses. Therefore, in some cases, economic considerations are needed to be made. The mutation testing's main problems are as follows:

a. Difficulty to find the equivalent mutant: Some modified versions are not destroyed with any data test. In other words, they have the same behavior as the main program, which the equivalent mutant has to be removed with accurate programming and no ambiguity [12].

b. Destroying the live mutants: Some mutants behave the same as the equivalent mutants due to the weakness of the data test. Therefore, they've got to be identified by choosing an appropriate data test [12].

c. High calculation expenses and high time-waste: In this method, we need to create several mutants using the mutation operators. Then the data test, operate themselves separately, in the next step, the number of killed and equivalent mutants for all the data test have to be operated and the mutation score has to be calculated. If the data test can't achieve the necessary mutation score, they have to improve and then the above steps have to be repeated several times [12].

III. Introducing How to Reduce The Mutation Test Expenses

These methods focus on reducing the calculation, time and the related expenses to improve the data test.

A. Reductions of Calculation and Time Expenses

1) "Do Fewer" Method: This method focuses reduction of number of the mutants and emphasizes on mutation operators that are more efficient instead of using all the operators. One of the most common "Do Fewer" methods is the Selective Mutation Tests [12], [13].

In this method, the mutation operators are divided in three

types of operator: Operand, Statement modifier operator, and the proposition modifier operator and according to the type of the under examination test, one or all the above categories could be selected.

2) "Do smarter" Method: This method emphasizes on smart use of the operators and the internal program states and one of the most important methods is the Weak Mutation Test [11].

The "Leonardo" Mutation System which is made based on this method, instead of running the mutants completely, analyzes them naturally and if the main program and mutants aren't the same, then decides to kill them. [14]

From the under comparison states, the Control Flow content, registers of the PC and the comparison results of the saved blocks could be named.

3) "Do faster" Method: This method focuses on faster running of the mutants, and one of the most common methods is the MSG (Mutation Base Schemata) [15], [16].

In this method, the mutation operators that have almost the same performance form one object. For instance, the AOR can cover all the mathematical operators lie "summation, reduction, multiplication and the division" The mutants made under such a circumstance are called "Meta Mutant" These programs are made in Compiler level so they have a high running speed.

4) Efficient Mutation Analyses: In this system, the human interference is significantly decreased; therefore the running speed is considerably high.

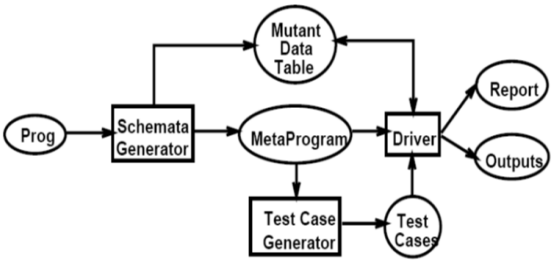

Fig. 2. The Efficient mutation analyses architect [9]

B. Expense Reductions Using the Appropriate Data Tests

One of the main principles of the Mutation Testing can be called as the appropriate data test selection. Otherwise, the mutation test system has to be run over and over.

1) The Genetic Algorithm [17]: In the optimizer section of the mutation test, Genetic Algorithms can be used to enhance the data test in order to do the enhancement automatically.

The Genetic Algorithm was introduced by "John Holland" [17] these algorithms are derivative from the Darwin theory using the gradual evolution in the nature. By a close look at the evolution process used by nature to solve its problems, we can come up with new and applicable ideas. Animals should adapt to their surrounding environment for their survival. The information taken from nature during many years is stored in chromosomes and in low levels in genes. Genetic algorithms automatically search in the input area for data with the maximum usage. For writing genetic algorithms, it is required to define a fitness function, capable of calculating the quality of chromosomes

Genetic Algorithms are described as follows:

a. The selection of random population from Chromosomes (Chromosomes in mutation testing are the data testing sets)

b. Calculating the fitness of each chromosomes (the fitness function in mutation testing is the Mutation Score function)

c. Creating a random population as:

- Two chromosomes which have the best fitness value are chosen, chromosomes with highest fitness value have the more chance to be selected.
- Chromosomes are chosen by probability.
- Children are mutated in each place by probability.

d. The new children replace the parents.

e. The Fitness Value of the new children is calculated and if the final condition is met, the algorithm is finished. Or else, it would go back to Step 3.

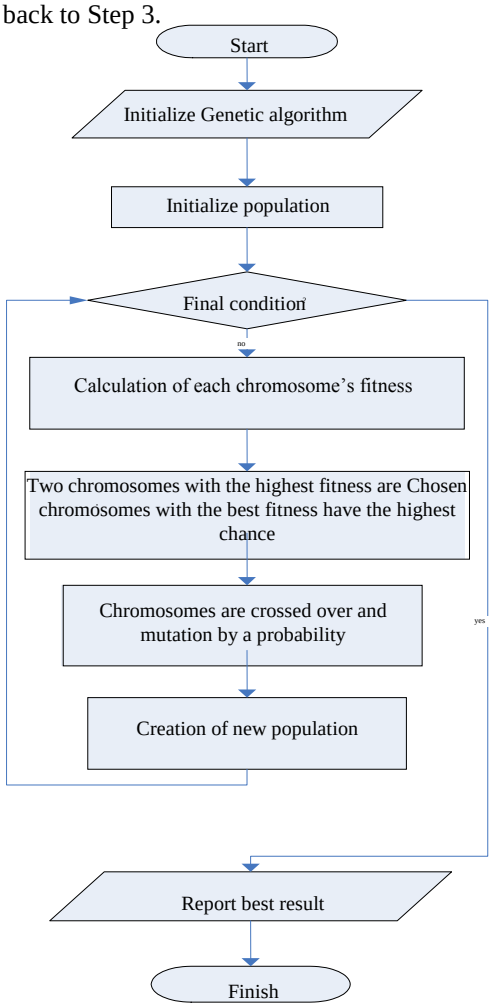

Fig. 3. Simple genetic algorithm [18]

Genetic algorithms use selection, combination and mutation genetic operators.

A. Selection: this process selects two chromosomes for combination and mutation operators. There are many methods for selecting parents such as random methods and methods related to fitness function [19]. In methods related to fitness function as opposed to random methods, since parents are selected for their performance, we can make sure that parents with higher fitness have more chance of being selected [18].

B. Combination: this gene operator, which is a series of bits (each bit represents a gene), make a new descendant from parent chromosomes chosen at random [20]. Its simplest way is 1-point cross over. In this method one point for combination of parents is chosen randomly, and then the exact

copy from the first parents to the combination point and the exact copy from the second parents to that point are selected. By this operator, different combinations of existing genes in the current population are made.

C. Mutation operator: after the combination process, mutation process is done, which means the prevention of gravitation of all the genetic population solutions toward a stable amount. Mutation randomly changes the new descendants. For example in binary codification, it is possible to change just a few numbers of bits from one to zero or vice versa randomly.

2) Using the bacteriological algorithm [17]: This algorithm is inspired from the reproduction of the Bacteria in the nature. In this model, as the Bacteria which are not combined in nature, we don't have the combination (cross over) operator to produce the new children and there's only a mutation operator.

In this algorithm, a threshold limit is introduced called BMS (Basic Mutation Score). Chromosomes with mutation scores lower than this limit have to undergo the mutation operator and then go to the next population. Chromosomes with points higher than the mutation score will go to the next population with no modification [16].

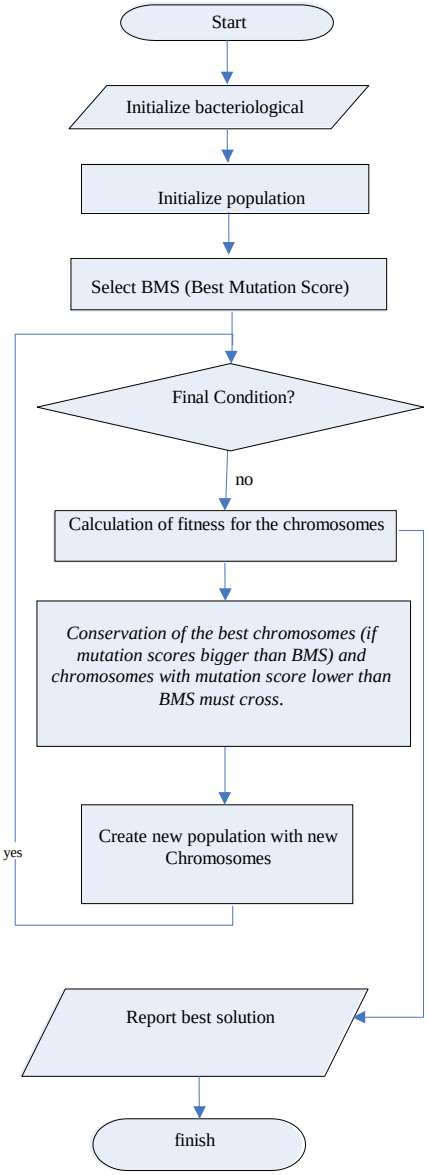

Fig. 4. Bacteriological algorithm [10]

The bacteriological algorithm is as follows:

a. Choosing the primary set of chromosomes
b. Calculating the fitness for each chromosome
c. Keeping the best chromosomes
d. Generation or selection of the best chromosomes
e. Mutations

In contrast with the genetic algorithm, the bacteriological models have more compatibility to store and keep the best chromosomes after the first reproduction from one reproduction to the next one. Convergence of the bacteriological algorithm in comparison to genetic algorithms is faster and in other words gets faster to the optimum answer, this ceases the calculation expenses related to combination operation are removed.

Removing of the combination operators is the major distinction with the genetic model. The other distinction is the necessity to keep a series of the best bacteria in memory which are remained from the previous population.

3) Using the optimum recommended algorithm [21]: This algorithm (fig 3&4) is a mixture of the normal genetic algorithm and the bacteriological algorithms.

In this algorithm, in each generation, if lacking any chromosome mutation scores, chromosomes mutated more and more, so that in this mutation get to the appropriate data, and after the parents get to the desired mutation score, they are combined to produce the best children.

In the recommended algorithm, the percentages of mutation rates and combination rates defined automatically and they'll be modified if necessary, it may be needed in the primary generation that the majority of the chromosomes, periodically, due to the low mutation score undergo the mutation operator. However as the mutation score enhances, in the final generation this need is less, however the cross over (combination) operation between the parents should end with a fine score to produce the appropriate children.

The Nc and Np parameters show the number of children and the amount of population. The value of Nc in the beginning of the process in Fig.2 is zero and with each operation the combination of the parents increases.

The BMS value, measures the modifiability of the chromosomes and using this value it could be guaranteed that chromosomes with higher mutation point than the BMS value, have the minimum requirement of going to the next generation.

In primary stages, thanks to high bearably of the mutation operator, a high diversity is observed in reproduction. As the BMS decreases in each generation, the number of parents that have to undergo the mutation operator will be less and the number those have to be combined increases. Therefore, the recommended algorithm has a good local search in final stages

IV. Implementation

In order to compare the methods of data enhancement, a mutation testing system is run which the optimizer section is the Genetic Algorithm, Bacteriological Algorithm and the recommended algorithm [21]. The mutation testing system

was written in Java and SQL Server is used to store and recover the data.

A. The Program in Use

The "TriType" software is used as a standard program and because of having the jumping and conditional types; it's used in majority of software tests. The input is three numbers as the sides of triangle and the output is 1 as a scalene, 2 as an isosceles, 3 as an equilateral or 4 as the sides don't meet the conditions to make a triangle. 282 Mutants are made according to this program.

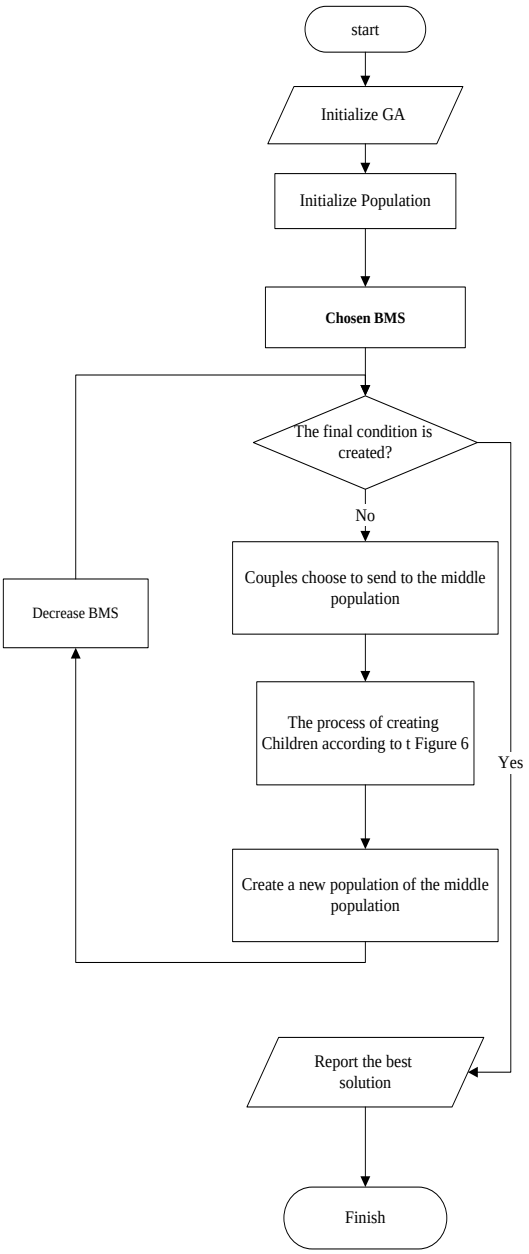

Fig. 5. Recommended algorithm

B. Choosing the Parents

Roulette wheel is used in order to choose the parents. Chromosomes in this method are put on a roulette wheel and according to the fitness values, occupy a position on the chromosome. It's obvious that the chromosomes with higher fitness values are more likely to be chosen.

For each side of the triangle, 10 bits are selected. In this simulation, because each chromosome is a data equivalent to 3 sides of the triangle, occupies a 30-bit space. Length of the population is constant as 6 chromosomes (data test).

C. Comparisons Between the Enhancement Methods in the Run Mutation Testing System

a. The number of the reproduced generations in recommended algorithm is less than other ways, because in early generations, the space of the input data is investigated enough using consecutive jumping and mutation [21].

b. The results show that, the time and cost of the recommended algorithm is less than other methods.

c. Because of the similarity of bacteriological algorithm and the random method, several rises and falls are observed and because each chromosome as an individual is optimizing and searching the problem space. We could claim that each of the last generation, are appropriate answers but they aren't desirable as group and therefore their mutation score is less than 90% [17].

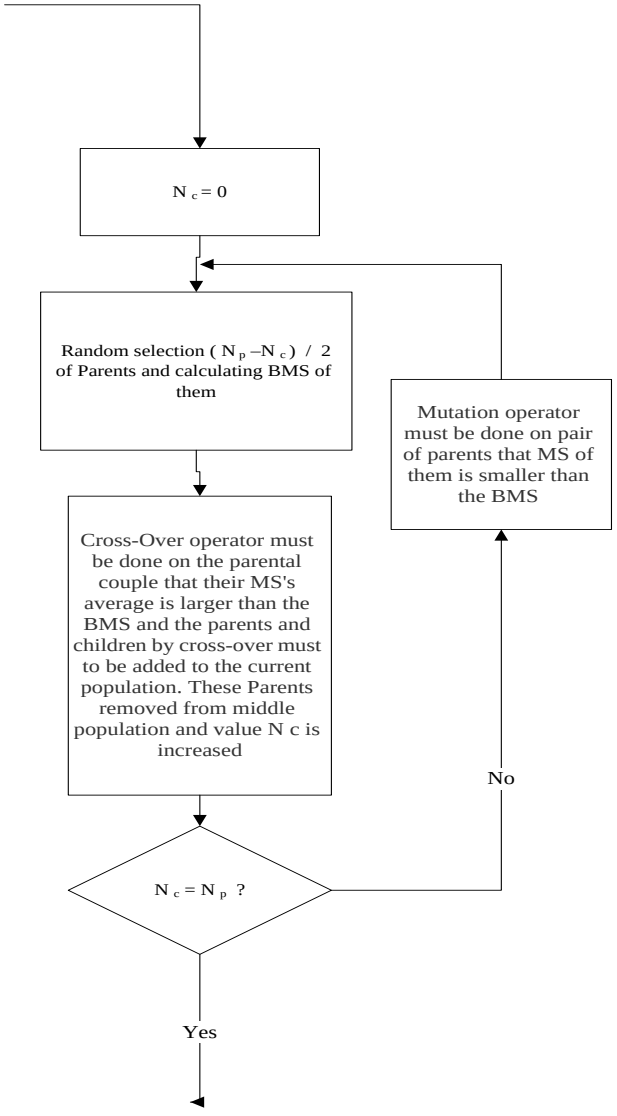

Fig. 6 . Process of figure 6 from recommended algorithm

V. Conclusion

The mutation testing is one of the software testing methods that are used to evaluate the software correctly. In this method, first the versions containing errors are from made the original

version and then the data test set will be operated on them and the results of these runs will be compared with the original software.

The result of these comparisons using special technics help us evaluate the data test sets and enhance them.

Mutation testing method has many advantages over the other testing methods, especially that in this method, the hidden errors which can't be detected in other methods will be detected automatically in this method and it is possible to automatically find the unknown errors which are not easily detected.

This method takes high time and cost because the testing data should be implemented on a lot of error contained versions, then the results are compared with the original program. Many methods have been invented to reduce these parameters which focus mainly on developing the running speed of faulty versions (mutant) and enhance quality of the data test set. In this article, it's been tried to introduce some of them.

REFERENCES

[1] J. C. Huang, "An Approach to Program Testing," *ACM Computing Surveys*, Vol.7, No. 3, September. 1975.

[2] B. Beizer, "Software Testing Techniques," Second Edition, *Van No strand Reinhold*, New York, 1990.

[3] G. J. Myers, "The Art of Software Testing," *John Wiley and Sons*, New York, 1979.

[4] K. Beck and E. Gamma, "Test-Infected: Programmers Love Writing Tests," *Java Report*, vol. 3, no 7, page. 37-50, 1998.

[5] A. Mathur, "Performance, effectiveness, and reliability issues in software testing, "in *Proc of the Fifteenth Annual International Computer Software and Applications Conference*, Tokyo, Japan, pages 604-605, September 1991.

[6] J. Offutt. "A Practical System for Mutation Testing: Help for the Common Programmer "in *Proc of the 12th International Conference on Testing Computer Software*, Test Conference, Pages. 99-109, 1995.

[7] R.A. DeMillo, D.S. Guindi, W.M. McCracken, A.J Offutt, K.N. King, and P. Univ, West Lafayette, IN ,"An extended overview of the Mothra software testing environment," *Software Testing, Verification, and Analysis*, 1988.

[8] R. A. DeMillo and E.H. Spafford, "The MOTHRA Software Testing Environment," in *Proc of the Eleventh Annual Software Engineering Workshop*, NASA, SEL-86-006, 1986.

[9] J. Offutt and R. Untch, "Mutation analysis using mutant schemata", in *Proc ISSTA '93 Proceedings of the 1993 ACM SIGSOFT international symposium on Software testing and analysis ACM New York*, NY, USA , Pages 139 – 148, 1993.

[10] B. Baudry, F. Fleurey, J. J ́ez ́equel, and Y. L. Traon," From genetic to bacteriological algorithms for mutation-based testing, Software Testing Verification and Reliability", *Software. Test. Verify. Reliably. Pages.73–96 published online 4th www.interscience.wiley.com*, January 2005.

[11] W. E. Howden, "Functional Programming Testing and Analysis, "*McGraw-Hill Book Company*, New York NY, 1987.

[12] A. T. Acree, T. A. Budd, R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Mutation analysis," *Technical, report GIT-ICS-79/08, School of Information and Computer Science*, Georgia Institute of Technology, Computer Science, Georgia Institute of Technology, Atlanta GA, September 1979.

[13] R. A. DeMillo and A. J. Offutt, "Experimental Results from an Automatic Test Case Generator," *ACM Transactions on Software Engineering Methodology*, vol. 2, no. 2, pp.109-175, April 1993.

[14] R. A. DeMillo, D. S. Guindi, K. N. King, W. M. McCracken, and A. J. Offutt, "An extended Overview of the Mothra software testing environment," in *Proceedings of the Second Workshop on Software Testing, Verification, and Analysis, Ban Alberta, IEEE Computer Society Press*, pages 142 – 151, July 1988.

[15] A. j. Offutt, "Investigations of the software testing coupling effect," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Volume 1, Issue 1, January 1992.

[16] A. Dana and T. A. Budd, "Two Notions of Correctness and Their Relation to Testing" *Act Informatics, pages31-45*, 1992.

[17] M.Fallah Rad, F. Akbari, and A. Javan Bakht. "Implementation of Common Genetic and Bacteriological Algorithms in Optimizing Testing Data in Mutation Testing," *cise*, 2010.

[18] D. E. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning", *Addison –Wesley*, 1989.

[19] R. Pargas, M. Harold, and R. Peck, "Test data generation using genetic algorithms," *Software testing, verification and reliability*, page 263-282, 1999.

[20] A. Watkins, "The automatic generation of software test data using genetic algorithms", in *proc of the fourth software quality conference, Dundee, Scotland*, pages 300, July 1995.

[21] M.Fallah Rad, M.i dehghan, and A. bakht, "optimization data sets in mutation testing with perfect genetic algorithm", isfs, 2008.

Mohsen Falah Rad was born at 1982 in Lahijan, Iran. He received B.S. Computer Engineering from Islamic Azad University, lahijan Branch, Iran, in summer 2004, and M.Sc degree in Computer software Engineering from Islamic Azad University, Tehran South Branch, Iran in summer 2007.

From 2011 till now, Manager of Applied Instruction Group, Islamic Azad University, lahijan Branch, Iran. 2008 until now, He has being faculty member of in Computer Engineering, Islamic Azad University, lahijan Branch, Iran (liau.ac.ir), 2008 until now, visiting fellow of "mehrastan" Non-government University, Astanih, Iran (mehrastan.ac.ir) and 2006 until now, visiting fellow of "sama" college, lahijan, Iran (lahijan-samacollege.ir). 2006 He was teaching at "Payamnoor" University and "sayf" high school, lahijan, Iran in 2005. He was published "Using modified genetic algorithm in Optimize segment at mutation testing ", Fallah-Rad, Dehqan takht fulady, Javanbakht, the second Congress on Intelligent Systems and Fuzzy Systems, Iran, 2008, - "Using fuzzy multiple criteria decision making In evaluation of software quality", Mohsen Fallah Rad, Farshad Akbari, Ahmad Javan Bakht." cise.2010, Wuhan, china, and "Implementation of Common Genetic and Bacteriological Algorithms in Optimizing Testing Data in Mutation Testing", Mohsen Fallah Rad, Farshad Akbari, Ahmad Javan Bakht." cise.2010, Wuhan, china and etc.

He has being Member of Iranian fuzzy association 2008 until now and Member of IACSIT 2011 till now. His main research interests include optimization algorithms and using them in software testing methods.

Mohadeseh Moosavi was born at 1985 in Mashhad, Iran. She received B.S. physic from Mashhad University, Iran, in summer 2007, and M.Sc degree in Nuclear physics from Guilan University, Guilan, Iran in winter 2011.

She was published "The investigation of fuel energy gain for poor-tririum fuels in fast ignition fusion", Mohadeseh Moosavi, Abbas Ghasemizad, Internationals conference in Nuclear energy for new Europe, (September2010), Slovenia, "The study of influences of decreased initial of tritium content in fuel capsule in fast ignition concept of inertial confinement fusion ", Mohadeseh Moosavi, Abbas Ghasemizad, advanced in applied physics and materials science congress, (May 2011), Turkey, "A review on using of quantum calculation techniques in optimization of data system of mutation test and its comparison with normal genetic algorithm and bacteriological.", Mohadeseh Moosavi, Mohsen falah rad, 3rd international conference on computer technology and develop (20011 ICCTD), China.

She has being Iran's membership in the Society of Physics (Student Category) and Iran's membership in the Society of Nuclear Physics from 2011 until now. Her main research interests include fusion technology and using quantum computing in computer science.