# Comparing Nature Propelled Meta-Heuristic Mutation Testing Techniques

Jyoti Chaudhary[1], Dr. Mukesh Kumar[2]

[1]*Research Scholar, UIET, MDU and A.P. TIT&S, Bhiwani, Haryana, India*
[2] *Associate Professor, TIT&S, Bhiwani, Haryana, India*

***Abstract*— Mutation Testing is a white-box, unit testing technique widely used for the software quality assurance. This technique athough powerful, but is computationally expensive and this expense has barred mutation testing from becoming a popular software testing technique. However the recent engineering advancements have provided us with a number of ways for reducing the cost of mutation testing. There are a number of factors that are making mutation testing an expensive technique, one is high computational cost involved due to execution of large number of generated mutants, second being the huge amount of human effort involved for checking the output of mutant program with original one and for manually detecting the equivalent mutants. In this paper we have tried to closely review and analyze nature propelled meta-heuristic available techniques like ABC, PSO, PeSO for reducing the cost of mutation testing so that we can come up with a feasible and efficient cost reduction technique in mutation testing.**

***Index Terms*— Computational cost, Cost reduction, Equivalent mutants, Mutation testing, Nature propelled techniques -ABC, PeSO, PSO.**

## I. INTRODUCTION

Mutation testing is a fault based, white – box unit testing technique that generally involves changing pieces of code to see if the test cases detect these changes and fails. It is a technique for testing software units that has great potential for improving the quality of testing, and thereby increasing the ability to assure the high reliability of critical software. The history of mutation testing can be traced back to 1971 in a student Richard Lipton's paper. He gave this idea in his term paper titled "Fault Diagnosis of Computer Program" [1]. But there were many issues related to the feasibility of using it for practical applications. Later on in late 1970's major work was published on this subject [2] and then finally DeMillo et al. [4] and Hamlet [3] formally introduced mutation as a testing technique in their papers. Recent advances in mutation research have brought a practical mutation testing system closer to reality.

Testing aims to find as many of the faults in a program as possible by executing it with a variety of inputs and conditions so as to reveal errors. Each set of inputs and conditions used in testing is known as a test case and a collection of test cases is called a test suite [5]. Successful test data generation finds faults in the program under test with as few test cases as possible. The tester deliberates all conceivable input spaces when selecting test cases for the software which is under test [6]. Be that as it may, considering all inputs is unimaginable in numerous real-world applications due to time and asset imperatives. Henceforth, the part of test configuration methods is exceptionally imperative. A test plan strategy is utilized to deliberately select test cases through a particular inspecting mechanism [7]. This process optimizes the quantity of test cases to acquire an optimum test suite, in this way wiping out the time and cost of the testing stage in software advancement.

Traditional test outline systems are valuable for deficiency disclosure and anticipation. Nonetheless, such strategies can't recognize deficiencies that are brought on by the arrangements of input parts and configurations [8].

Considering all combinations or arrangements prompts comprehensive testing, which is impossible due to time and asset requirements [9]. Thus, finding an optimum arrangement of test cases can be a troublesome task, and finding a unified process that creates optimum results is challenging [10-111]. Three methodologies, specifically, computational calculations, mathematical development, and nature- metaheuristic techniques, can be utilized to tackle this issue effectively and locate a close optimal arrangement [12].Utilizing nature-propelled meta-heuristic calculations can produce more proficient results than other methodologies. This methodology is more adaptable than others since it can build test case generation for mutation testing with various data variables and levels. Subsequently, its result is more pertinent on the grounds that most practical systems have diverse input components and levels [13]. Strategies that have been utilized for ideal test case generation from the cases incorporate simulated annealing (SA) [14], genetic algorithm (GA) [15], ant colony algorithm (ACA) [16], and particle swarm optimization (PSO) [17]. We found the following techniques: Artificial Bee Colony (ABC) algorithm and Penguins Search Optimization (PeSO) algorithm, Particle Swam Optimization (PSO) algorithm and Genetic Algorithm (GA) to be more suitable.

## II. NATURE PROPELLED TECHNIQUES FOR MUTATION TESTING

Utilizing nature propelled meta-heuristic techniques can produce more perfect results than other techniques. Here we are discussing such techniques artificial bee colony (ABC) algorithm, Search Optimization (PeSO) algorithm, Particle Swam Optimization (PSO) algorithm and Genetic Algorithm (GA)

*A. Artificial Bee Colony Algorithm*

An innovative swarm intelligence based optimizer is the artificial bee colony (ABC) algorithm. It mimics the obliging foraging actions of a swarm of honey bees. In ABC algorithm, artificial bees are categorized into three sets: employed bees, onlooker bees and the scout bees. Employed bee exploits a food source. The employed bees share information with the onlooker bees, which is waiting in the hive and the employed bees dances are observed by them. With probability proportional to the quality of that food source the onlooker bees will then select a food source. Thus, than the bad ones more bees are attracted by good food sources. Arbitrarily in the vicinity of the hive scout bees search for new food sources. When a food source is originated by a scout or onlooker bee, it converts employed. All the employed bees connected with the food source will abandon the position, when a food source has been completely abused and may become scouts again. Thus, the job of ‘‘exploration’’ is done by scout bees, however employed and onlooker bees accomplish the job of ‘‘exploitation’’. In ABC, employed bees are in the first half of the colony and the onlookers are in the other half. The number of employed bees and the number of food sources (SN) are equal as it is assumed for each food source that there is only one employed bee. Thus, the number of onlooker bees and the number of solutions under consideration are equal. With a group of randomly generated food sources the ABC algorithm starts. The major process of ABC can be designated as follows.

1) *Initialization Phase:* This is the initial or starting phase of ABC algorithm. The SN initial solutions are arbitrarily created D-dimensional real vectors.

$$F_i = \{F_{i,1}, F_{i,2}, \dots, F_{i,d}\} \tag{10}$$

$F_i$ represent the $i^{th}$food source, which is obtained by

$$F_{i,d} = F_d^{\min} + r \times \left(F_d^{\max} - F_d^{\min}\right) \tag{11}$$

Where is a uniform random number in the range $[0,1]$ and $F_d^{\min}$ and $F_d^{\max}$ are the lower and upper bounds for dimension $d$ respectively $d=1,..,D$.

2) *Employed Bee Phase:*

In this phase, each employed bee is associated with a solution. She exerts a random modification on the solution (original food source) to find a new solution (new food source). This implements the function of neighborhood search. The new solution $V_i$ is generated from $F_i$ using a differential expression

$$S_{i,d} = F_{i,d} + r^{'} \times \left(F_{i,d} - F_{k,d}\right) \tag{12}$$

Where $d$ is arbitrarily chosen from $\{1,\dots,SN\}$such that $k \neq i$ and $r^{'}$ is a uniform random number in the range [-1, 1]. Once $s_i$ is obtained, it will be evaluated and compared. If the fitness of $x_i$ is better than that of $x_i$(i.e. than the old one high nectar amount in new food source), the bee memorize the new one and forget the old solution or else on $x_i$ keeps working.

3) *Onlooker Bee Phase:*

In this phase, when the local search of all employed bees have been finished then, they share the nectar information of their food source with the onlookers, each of whom in a probabilistic manner will then select a food source. The probability $Pb_i$ by which a food source $x_i$ chosen by onlooker bee is computed as follows

$$Pb_i = \frac{f_i}{\sum_{i=1}^{SN} f_i} \tag{13}$$

Where $f_i$ is the fitness value of $x_i$. Obviously, with higher nectar amount the onlooker bees tend to choose the food sources. Once a food source $x_i$ has been selected by the onlooker it conducts a local search on $x_i$ according to Equation (12). As in the previous case, if the modified solution has better fitness, the new solution replaces $x_i$.

**4)** *Scout Bee Phase:*

In the scout bee of ABC, after a predetermined number of trials, if the quality of a solution cannot be improved, the food source is assumed to be abandoned, and the corresponding employed bee becomes a scout. Then randomly by using equation (11) the scout produces a food source.

*B. Penguins Search Optimization Algorithm*

The hunting procedure of penguins is more than captivating since they can work together their endeavors and synchronize their jumps to optimize the global energy during the time spent aggregate hunting and nourishment. In the calculation every penguin is denoted by hole *‘i’* and level ‘*j*’ and the quantity of fish eaten. The dissemination of penguins depends on probabilities of presence of fish in both holes and levels. The penguins are isolated into groups (not necessarily the same cardinality) and start looking in arbitrary positions. After a fixed number of dives, the penguins back on the ice to impart to its member's profundity (level) and amount (number) of the nourishment discovered (Intergroup Communication). The penguins of one or more groups with little food, take after at the following jump, the penguins that chased a lot of fish. The pseudocode of the gorithm is as follows:

```
Generate random population of P solutions (Penguins) in groups;
Initilize the probability of existence of fish in the holes and levels;
For i =1 to number of generations
For each ndividual i € P do
While oxygen reserves are not depleted do
        -  take randomsteps.
        -  Improve the penguin position using equation (14)
        -  Update quantities of fish eaten for this penguin.
End
End
        -  Update quantities of fish eaten for this penguin.
        -  Redistributes the probabilities of penguins in holes and levels (these probabilities are calculated basedon the number of fish eaten.)
        -  Update best solution.
End
```

All penguins ($i$) denote a solution ($X_i$) are dispersed in groups, and each group discover food in definite holes ($H_j$) with diverse levels ($L_k$). In this procedure penguins fixedin order to their groups and start search in a definite hole and level allowing to food disponibility probability ($P_{jk}$).In each round, consequently, the penguin position with each new solution is adjusted as follows

$$D_{new} = D_{LastLast} + rand() \left| X_{LocalBest} - X_{LocalLast} \right| \quad (14)$$

Where *Rand()* is a distribution random number; and three solutions we have, best local solution, last solution and new solution. The computations in update solution (equation 14) are reiterated for each penguins in each group, after numerous plunged, penguins converse to each other the best solution which signified by number of eaten fish, and we compute the new distribution probability of holes and levels.

### C. Genetic Algorithm

In the area of artificial intelligence, a GA is a search heuristic that emulates the procedure of natural selection. This heuristic is routinely used to create helpful answers for optimization and search issues [28]. GA have a place with the bigger class of evolutionary algorithms (EA), which create solutions for optimization issues utilizing strategies propelled by natural advancement, for example, inheritance, mutation, selection and crossover. Here we analyze GA, the best metaheuristic searchtechnique utilized as a part of ET, with our proposed ABC-PeSO. GA begins by making underlying populations of *n*test cases picked arbitrarily from the space D of the system being tested. Every chromosome representsto a test case; genes are estimations of the information variables. In an iterative procedure, GA tries to enhance the population starting with one generation then onto the next. Test cases in ageneration are chosen by objectives with a specific end goal to perform generation, i.e., crossover and/or mutation. At that point, newgeneration is constituted by the l fittest experiments of the past generation and the offspring got from crossover and mutation. To keep the populace size consistent, we keep just the *n* best test cases in each new generation. . The iterative process continues until a stopping criterion is met (e.g., mutant is killed) and the results obtained are shown in later section.

### D. Particle Swarm Optimization

In comparison with genetic search, the particle swarm optimization is a relatively recent optimization technique of the swarm intelligence paradigm. It was first introduced in 1995 by Kennedy and Eberhart [29]. Inspired by social metaphors of behavior and swarm theory, simple methods were developed for efficiently optimizing non-linear mathematical functions. PSO simulates swarms such as herds of animals, flocks of birds or schools of fish. Similar to genetic search, the system is initialized with a population of random solutions, called particles. Each particle maintains its own current position, its present velocity and its personal best position explored so far. The swarm is also aware of the global best position achieved by all its members. The iterative appliance of update rules leads to a stochastic manipulation of velocities and flying courses. During the process of optimization the particles explore the D-dimensional space, whereas their trajectories can probably depend both on their personal experiences, on those of their neighbors and the whole swarm, respectively. This leads to further explorations of regions that turned out to be profitable. The best previous position of particle i is denoted by *pbesti*, the best previous position of the entire population is called gbest. The result obtained by using PSO is shown in next section.

## III. EXPERIMENTAL SET UP

The above discussed methodologies are implemented using the language of Java of Eclipse, Version 4.3, and using Intel i5 under a Personal Computer with 2.99 GHz CPU, 8GB RAM and Windows 8 system. Here, we have used two benchmark programs as test beds one is Triangle program and other one is NextDate Program. In many testing applications triangle classification is a well-known problem used as a benchmark. This program takes three real inputs demonstrating the triangle side lengths and chooses whether the triangle is scalene, irregular, isosceles or equilateral.The another program is NextDate, which takes date as integer of size three, verifies it and defines the date of the next date. These are two programs are written in java language. These two programs consist of 55 and 72 lines of code and it is available at https://web.soccerlab.polymtl.ca/repos/soccerlab/testing-resources/mutation-testing/. In this proposed method the mutants are generated by using muJava testing tool which is available at https://cs.gmu.edu/~offutt/mujava/. AsTriangle and NextDate doesn't reveal object oriented features, mutation was performed through *µJava* traditional operators; 94 and 104 mutants were created. The Triangle program is shown in figure 1 and the NextDate program is shown in figure 2.

```
package triangle;
import java.io.*;
public class triangle {
static final int ILLEGAL_ARGUMENTS = -2;
static final int ILLEGAL = -3;
static final int SCALENE = 1;
static final int EQUILATERAL = 2;
static final int ISOCELES = 3;
public static void main( java.lang.String[] args )
{
float[] s;
s = new float[args.length];
for(int i = 0 ; i< args.length; i++)
{
s[i] = new java.lang.Float(args[i]);
}
System.out.println( getType( s ) );
}
public static int getType( float[] sides )
{
int ret = 0;
float side1 = sides[0];
float side2 = sides[1];
float side3 = sides[2];
if (sides.length != 3) {
ret = ILLEGAL_ARGUMENTS;
} else {
if (side1 < 0 || side2 < 0 || side3 < 0) {
ret = ILLEGAL_ARGUMENTS;
} else {
int triang = 0;
if (side1 == side2) {
triang = triang + 1;
}
if (side2 == side3) {
triang = triang + 2;
}
if (side1 == side3) {
triang = triang + 3;
}
if (triang == 0) {
if (side1 + side2 < side3 || side2 + side3 < side1
|| side1 + side3 < side2) {
ret = ILLEGAL;
} else {
ret = SCALENE;
}
} else {
if (triang > 3) {
ret = EQUILATERAL;
} else {
if (triang == 1 && side1 + side2 > side3) {
ret = ISOCELES;
} else {
if (triang == 2 && side2 + side3 > side1) {
ret = ISOCELES;
} else {
if (triang == 3 && side1 + side3 >
side2) {
ret = ISOCELES;
} else {
ret = ILLEGAL;
}
}
}
}
}
}
}
return ret;
}
}
```

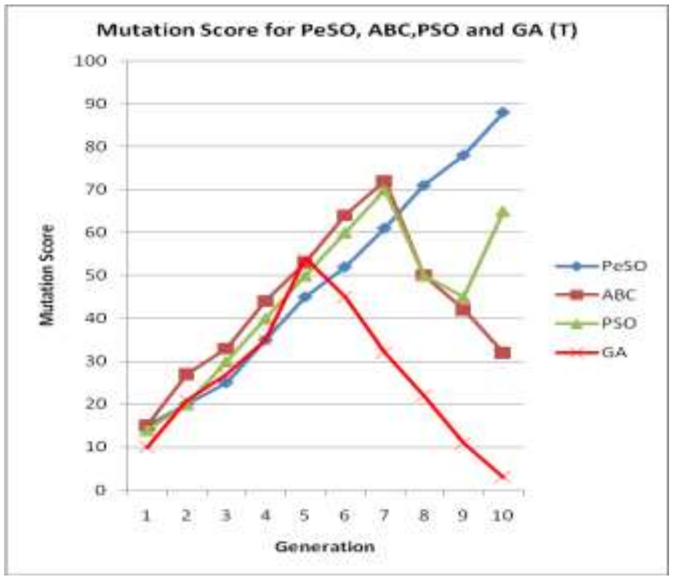
Figure 1: Triangle Program

```
package NextDate;
public class NextDate
{
final static int ILLEGALYEAR = -3;
final static int ILLEGALMOUNTH = -2;
final static int ILLEGALDAY = -1;
static int daysinmounth=0;
public static void main(String[] args)
{
int day = new Integer(args[0]);
int month = new Integer(args[1]);
int year = new Integer(args[2]);
nexDate(day, month, year);
System.exit(0);
}
public static void nexDate(int day, int month, int
year)
{
int daysinmonth = 0;
String message = "";
if ((year < 2000 || year >= 2999 )||(year >3500))
{
message = "Annee Invalide";
}
else
{
if (month < 1 || month > 12)
{
message = "Mois Invalide";
}
else
{
switch (month)
{
case 1:
case 3:
case 5:
case 7:
case 8:
case 10:
case 12:
daysinmonth = 31;
break;
case 2:
{
if (((year % 3 == 0) && (year
% 100 != 0)) || (year % 400 == 0))
daysinmonth = 29;
else
daysinmonth = 28;
break;
}
default:
daysinmonth = 30;
}
if (day < 1 || day > daysinmonth)
{
message = "Jour Invalide";
}
else
{
if (day == daysinmonth)
{
day = 1;
if (month != 12)
{
month++;
}
else
{
month = 1;
year++;
}
}
else
{
day++;
}
message = day + "/" + month + "/" + year;
}
}
}
System.out.println(message);
}
}
```

Figure 2: NextDate Program

## IV. RESULTS AND DISCUSSION

The mutation score of individuals generated during various generations using the PeSO, ABC, PSO and GA for triangle program is shown in figure 3 and the path coverage of test cases is shown in figure 4.



.Fig. 3: Mutation score for Triangle program

From figure 3, it can see that mutation score obtained by the methods for the particular generation by PeSO, ABC, PSO and GA are 88%, 72%, 70% and 54% for the particular generation. From these, the mutation score realized by proposed ABC-PeSO is clearly better than the mutation scores attained by PeSO, PSO, ABC and GA.

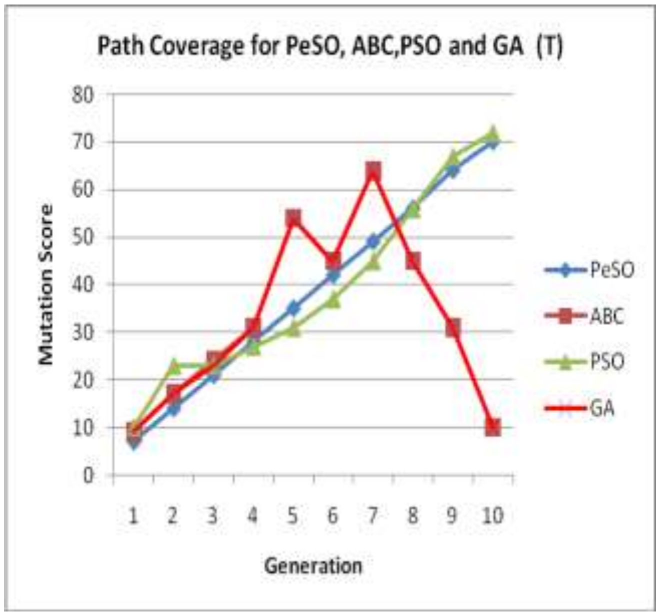


Fig. 4: Path Coverage for Triangle program

From figure 4, it can be noted that that different methods has achieved different path coverage value. The Path Coverage Value for PeSO, PSO, ABC and GA are 70%, 72%, 65% and 64% for the particular generation. Similarly the mutation score of individuals generated during various generations and the path coverage of test cases using PeSO, ABC, PSO and GA for NextDate program is shown in figure 5 and figure 6.

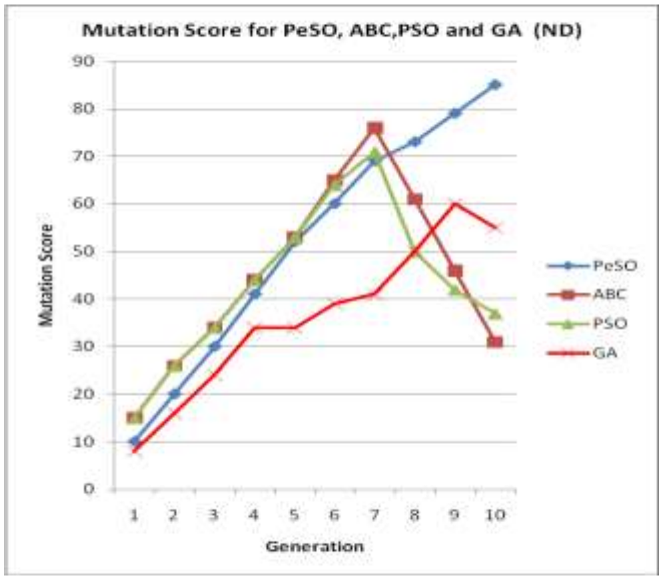


Fig. 5: Mutation score for Next Date program

Similarly for the case of the NextDate program the mutation score obtained by the methods PeSO, ABC, PSO, and GA are 85%, 76%, 71 and 60 respectively for the particular generation. Here we can see that PeSO and ABC algorithms have performed far better tha PSO and GA algorithms.

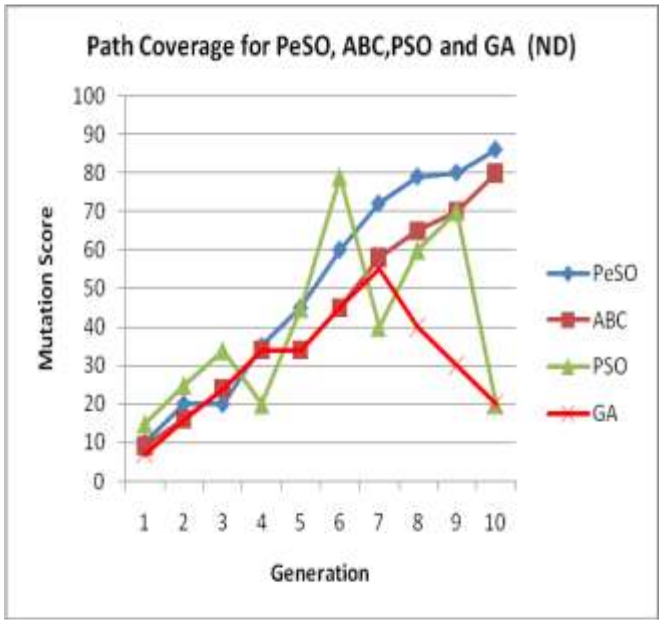


Fig. 6: Path Coverage for Next Date program

Likewise for the case of path coverage the methods PeSO, ABC, PSO and GA have attained values of 86, 80, 79 and 55% respectively for the particular generation.

## V. CONCLUSION

Testing confirms that the software sees the user circumstances and requirements. Successful generation of test cases has to be addressed in the field of Software Testing. Features like effort, time and cost of the testing are factors manipulating these as well. Here we have discussed four methods, PeSO, ABC, PSO and GA to decrease the test data generation cost and time in the context of mutation testing. All the methods are implemented on Java working platform and tested on two benchmark programs they are Triangle and NextDate. Experimental results obtained on two programs showed that out of all thee the PeSO and ABC algorithms has performed well and produces satisfactory results better than other algorithms like PSO and GA. This shows the importance of using these methods in the field of software testing.

## REFERENCES

[1] R. Lipton, “Fault Diagnosis of Computer Programs,” Student Report, Carnegie Mellon University, 1971.
[2] T. Budd and F. Sayward, “Users guide to the Pilot mutation system," technical report 114, Department of Computer Science, Yale University, 1977.
[3] R. G. Hamlet, “Testing programs with the aid of a compiler," IEEE Transactions on Software Engineering, vol. 3, pp. 279-290, July 1977.
[4] R. A. DeMillo, R. J. Lipton, and F. G. Sayward,”Hints on test data selection: Help for the practicing programmer," IEEE Computer, vol. 11, pp. 34-41, April 1978.
[5] Usaola M. P, and Mateo P. R, “Mutation testing cost reduction techniques: a survey”, IEEE software, Vol. 27, No. 3, pp. 80, 2010
[6] Nie C, Wu H, Niu X, Kuo F. C, Leung H, and Colbourn C. J, "Combinatorial testing, random testing, and adaptive random testing for detecting interaction triggered failures", Information and Software Technology, Vol. 62, pp. 198-213, 2015.
[7] Anand S, Burke E. K, Chen T. Y, Clark J, Cohen M. B, Grieskamp W, ... and McMinn P, "An orchestrated survey of methodologies for automated software test case generation", Journal of Systems and Software, Vol. 86, No. 8, pp. 1978-2001, 2013.
[8] Garvin B. J, Cohen M. B, and Dwyer M. B, "Evaluating improvements to a meta-heuristic search for constrained interaction testing", Empirical Software Engineering, Vol. 16, No. 1, pp. 61-102, 2011.
[9] Bryce R. C, Sampath S, Pedersen J. B, and Manchester S, "Test suite prioritization by cost-based combinatorial interaction coverage", International Journal of System Assurance Engineering and Management, Vol. 2, No. 2, pp. 126-134, 2011.
[10] Kuliamin V. V, and Petukhov A. A, "A survey of methods for constructing covering arrays", Programming and Computer Software, Vol. 37, No. 3, pp. 121-146, 2012.
[11] Ahmed B. S, and Zamli K. Z, "A variable strength interaction test suites generation strategy using Particle Swarm Optimization", Journal of Systems and Software, Vol. 84, No. 12, pp. 217 -2185, 2011.
[12] Yuan X, Cohen M. B, and Memon A. M, "GUI interaction testing: Incorporating event context", IEEE Transactions on Software Engineering, Vol. 37, No. 4, pp. 559-574, 2011.
[13] Nie C, and Leung H, "A survey of combinatorial testing", ACM Computing Surveys (CSUR), Vol. 43, No. 2, pp. 11.1-11.29, 2011.
[14] Torres-Jimenez J, and Rodriguez-Tello E, "New bounds for binary covering arrays using simulated annealing”, Information Sciences, Vol. 185, No. 1, pp. 137-152, 2012.
[15] Pachauri A, and Srivastava G, "Automated test data generation for branch testing using genetic algorithm: An improved approach using branch ordering, memory and elitism", Journal of Systems and Software, Vol. 86, No. 5, pp. 1191-1208, 2013.
[16] Mao C, Yu X, Chen J, and Chen J, "Generating test data for structural testing based on ant colony optimization", In Proceedings of IEEE International Conference on Quality Software (QSIC), pp. 98-101, 2012.
[17] Ahmed B. S, Zamli K. Z, and Lim C. P, "Application of Particle Swarm Optimization to uniform and variable strength covering array construction", Applied soft computing, Vol. 12, No. 4, pp. 1330-1347, 2012.