Journal of Engineering and Applied Sciences 12 (Special Issue 3): 6635-6648, 2017
ISSN: 1816-949X

Optimal Test Case Generation in Mutation Testing-A Hybrid Artificial Bee Colony-Penguin Search Optimization (ABC-PeSO) Approach

[1]Jyoti Chaudhary and [2]Mukesh Kumar
[1]Department of Computer Engineering,
[2]The Technological Institute of Textile and Sciences, Birla Colony, Bhiwani,
127021 Haryana, India

Abstract: Software development associations spend extensive part of their financial plan and time in testing related exercises. The adequacy of the verification and validation process relies on the number of errors found and corrected before releasing the software to the customer side by means of testing procedure. Mutation testing is a fault-based programming testing procedure that has been broadly concentrated on for more than three decades. As of late, evolutionary or optimization algorithms have been demonstrated reasonable for decreasing the cost of test case generation in the context of mutation testing. In this study, we develop a new combination of optimization algorithms called a hybrid Artificial Bee Colony-Penguin Search Optimization (ABC-PeSO) approach for generating efficient test input data in the context of mutation testing to decrease the cost of such a test scheme. Here, in this hybrid method, the ABC algorithm is enhanced by converting the random search process of scout bee phase to randomized process by using PeSO. Also, to evaluate the performance of the proposed ABC-PeSO, a detail comparison is conducted for different algorithms like ABC, PeSO, Genetic Algorithm (GA) and Particle Swarm Optimization (PSO). The comparison between the proposed and existing method in behalf of two benchmark programs as mutation score and path coverage. According to the analysis, it is concluded that proposed ABC-PeSO approach has produced better results.

Key words: Mutation testing, test case generation, Artificial Bee Colony (ABC), Penguin Search Optimization (PeSO), hybridization, path coverage

INTRODUCTION

The objective of testing is to reveal, however many faults as could be expected under the circumstances with an arrangement of test sets. Software associations spend more than 40-50% of their advancement cost in programming testing (Jia and Harman, 2011). So, as to test programming, test data must be produced. Producing test data physically is moderate, costly and requires thorough endeavors. Thus, automated test data generation techniques can be utilized to facilitate the procedure and reduce the cost. Testing can be fall into two categories: black box testing and white box testing (Nidhra and Dondeti, 2012). Black box testing is mostly a validation system that verifies whether the functionality of an application meets the specifications based on the customer requirements. In any case, white box testing is a verification system that examines the program structure and derives test data from the program logic/code (Nayak and Mohapatra, 2010). Mutation testing is a category of white box testing. Fundamentally, it is fault based testing situated in light of mutation analyses which overcome the constraints of other testing approaches. Mutation analysis recognizes method to change, i.e., to adjust, software qualities. Mutation testing gives a testing rule which can be utilized to measure the adequacy of a test set or data as far as its capacity to distinguish faults (Fraser and Zeller, 2012).

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 (Usaola and Mateo, 2010). 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 (Nie *et al*., 2015). Be that as it may, considering all inputs is unbelievable in numerous real-world applications due to time and resource goals.

Corresponding Author: Jyoti Chaudhary, Department of Computer Engineering,
The Technological Institute of Textile and Sciences, Birla Colony, Bhiwani, 127021 Haryana, India

Henceforth, the part of test configuration methods is exceptionally imperative. A test plan is used to intentionally choose test cases through a particular inspecting mechanism (Anand *et al*., 2013). 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 development. Various studies have proposed different functional test designs for example, equality class dividing, boundary value examination and circumstances and effect investigation by means of decision tables (Bansal, 2014).

In general, the tester objective is to utilize more than one testing technique on the grounds that distinctive issues might be identified when diverse testing strategies are utilized (Jia and Harman, 2009). Be that as it may with the inconceivable development and improvement of software systems and their configurations, the probability of the event of issues has expanded due to the arrangements of these configurations, especially for exceedingly configurable software systems (Ahmed *et al*., 2014). 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 (Garvin *et al*., 2011). Considering all combinations or arrangements prompts comprehensive testing which is impossible due to time and asset requirements (Bryce *et al*., 2011).

Considering all combinations in a minimized test suite is a hard computational optimization issue in light of the fact that scanning for the ideal set is a NP-hard issue. Thus, finding an optimum arrangement of test cases can be a troublesome task and finding a unified process that creates optimum results is challenging (Kuliamin and Petukhov, 2011; Ahmed and Zamli, 2011). Three methodologies, specifically, computational calculations, mathematical development and nature-met heuristic techniques can be utilized to tackle this issue effectively and locate a close optimal arrangement (Yuan *et al*., 2011). Utilizing nature-propelled met 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 (Nie and Leung, 2011). Strategies that have been utilized for ideal test case generation from the cases incorporate Simulated Annealing (SA) (Torres-Jimenez and Rodriguez-Tello, 2012), Genetic Algorithm (GA) (Pachauri and Srivastava, 2013). Ant Colony Algorithm (ACA) (Mao *et al*., 2012) and Particle Swarm Optimization (PSO) (Ahmed *et al*., 2012). In this study, we have presented a novel hybrid Artificial Bee Colony-Penguin Search Optimization (ABC-PeSO) for optimal test case generation on basis of mutation score and path Coverage which are utilized as the test suitability measures to obtain resourceful test casesin the context of mutation testing.

Literature review: Baker and Habli (2013) provided an experiential estimation of the application of mutation testing to Airborne Software systems which already satisfied the coverage requirements for certification. Specifically, they applied mutation testing to safety critical software developed using high-integrity subsets of C and Ada, identified the most effective mutant types and analyzed the root causes of failures in test cases. Their findings showed how mutation testing could be effective where traditional structural coverage analysis and manual peer review have failed. They also showed that several testing issues begin beyond the test activity and this suggested improvements to the requirements definition and coding process. Their study also examined the relationship between program characteristics and mutation survival and considered about the program size provided a means for targeting test areas most likely to have dormant faults. Industry feedback was also provided, particularly about the mutation testing can be integrated into a typical verification life cycle of Airborne Software.

Fraser and Arcuri (2015) extended and evaluated the whole test suite generation approach for mutation testing. In previous work, the whole test suite approach led to large improvements in performance for branch coverage. The presence of infeasible testing targets does not harm the search which is the simple reason for large improvement. That paper confirmed that this was also the case for mutation testing, by performing an empirical study on 100 Java projects randomly selected from Source Forge, i.e., the SF100 corpus (consisting of 8, 963 classes, for a total of more than two million lines of code). Besides the whole test suite approach, evosuite also included several novel optimizations for mutation testing such as the use of infection conditions, optimized mutation operators and prioritized test execution. Their results showed that using standard mutation testing in test case generation would not scale up to the complexity of real-world software.

Debroy and Wong (2014) have proposed a strategy for automatically fixing faults in a program by combining the ideas of mutation and fault localization. Statements ranked in order of their likelihood of containing faults are mutated in the same order to produce potential fixes for

the faulty program. The strategy was evaluated using 8 mutant operators against 19 programs each with multiple faulty versions. Their results indicated that 20.70% of the faults are fixed using selected mutant operators, suggesting that the strategy holds merit for automatically fixing faults. The impact of fault localization on efficiency of the overall fault-fixing process was investigated by experimenting with two different techniques, Tarantula and Ochiai, the latter of which has been reported to be better at fault localization than Tarantula and also proved to be better in the context of fault-fixing using their strategy.

Belli *et al.* (2016) have introduced the concept of Model-Based Mutation Testing (MBMT) and position it in the landscape of mutation testing. Two elementary mutation operators insertion and omission are exemplarily applied to a hierarchy of graph-based models of increasing expressive power including directed graphs, event sequence graphs, finite state machines and state charts. Test cases generated based on the mutated models (mutants) are used to determine not only whether each mutant can be killed but also whether there are any faults in the corresponding System Under Consideration (SUC) developed based on the original model. Novelties of their approach are: evaluation of the fault detection capability (in terms of revealing faults in the SUC) of test sets generated based on the mutated models and superseding of the great variety of existing mutation operators by iterations and combinations of the two proposed elementary operators. Three case studies were conducted on industrial and commercial real-life systems and demonstrated the feasibility of MBMT approach in detecting faults in SUC and analyzed its characteristic features.

Habibi and Mirian-Hosseinabadi (2015) have introduced a new six-stage testing procedure for event driven web applications to overcome EDS testing challenges. The stages of the testing procedure include dividing the application based on its structure, creating functional graphs for each section, creating mutants from functional graphs, choosing coverage criteria to produce test paths, merging event sequences to make longer ones and deriving and running test cases. They have analyzed their testing procedure with the help of four metrics consisting of Fault Detection Density (FDD), Fault Detection Effectiveness (FDE), Mutation Score and Unique Fault. Using that procedure, they have prepared prioritized test cases and also discovered a list of unique faults by running the suggested test cases on a sample real-world web application called academic e-mail system.

MATERIALS AND METHODS

A hybrid Artificial Bee Colony-Penguin Search Optimization (ABC-PeSO) approach: Mutation testing is utilized as fault based testing to overcome constraints of other testing approaches yet, it is recognized as costly process. In mutation testing, a good test case is one that kills one or more mutants by delivering different mutant yield from the original program. In order to select or generate a good test case optimization algorithms have been demonstrated its suitability for generating an optimal test cases as well as reducing the cost of data generation in various testing approaches. The objective of the proposed methodology is to reduce the cost of mutation testing, create the optimal test cases that disclose faults and kill mutants. In this proposed methodology, we develop a new combination of optimization algorithms called a hybrid Artificial Bee Colony-Penguin Search Optimization (ABC-PeSO) approach for generating efficient test input data in the context of mutation testing to decrease the cost of such a test scheme. In this proposed method, we will use two benchmark programs called triangle and next date and these two programs are written in Java. Here, Jester/mu Java testing tool is used to create mutants for the software under test. In this proposed methodology, mutation score and path coverage are utilized as the test suitability measures to obtain resourceful test cases. The aim is to create the test cases with good quality and express the number of errors from the software during the testing with low cost and less time. Here, a novel hybrid Artificial Bee Colony-Penguin Search Optimization (ABC-PeSO) algorithm is designed for test case optimization which works as a tester. In ABC algorithm on scout bee phase, if the quality of a solution cannot be enhanced after a predetermined number (limit) of trials, the food source is assumed to be abandoned and the equivalent employed bee becomes a scout. The scout will then produce a food source randomly and in order to improve the optimization performance of scout bee phase, penguin search optimization is utilized here. The proposed method is implemented on Java platform and the architecture of the proposed method is shown in Fig. 1. The architecture of

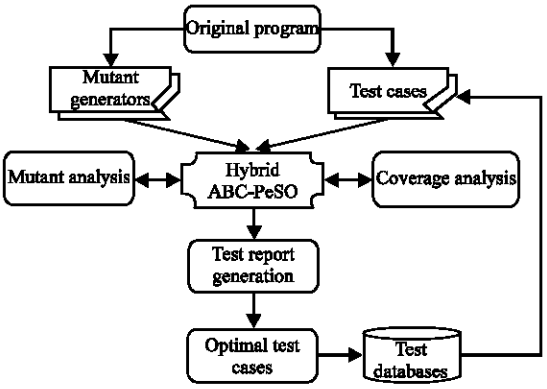

Fig. 1: Architecture of proposed method

proposed method is shown in Fig. 1 utilizes hybrid ABC-PeSO based optimal test case selection in mutation testing. The architecture has the subsequent actions:

- Read the source code or original program
- Mutant generation and test cases extraction
- Mutation analysis and coverage analysis based on metrics of mutation score and test path
- Generate the optimal test cases form the initial test cases using hybrid ABC-PeSO
- Resultant thst cases are stored inside the test databases

At start with read the source code or original program, this is to be considered under the mutation testing. The initial set of test sequences and test cases are generated from the original code by means of manual entry or random generations. Here, the hybrid ABC-PeSO which takes the input source code or original code. The mutated versions of the original code are generated by mutant generator. On basis of analysis of mutation score and path coverage based on the values obtained the optimal test cases are generated by the hybrid ABC-PeSO. The generated optimal test cases are finally stored in the test databases. The explanation of the proposed method is given in this study that we give the indication about mutation testing.

Mutation testing overview: In the mutation testing, a fault is presented by a small adjustment of a correct program code and it was initially proposed by Hamlet (1977). The adjusted program is called mutant, it is consider being killed mutant, if the test case recognizes different between mutant and the first program. The mutant is called alive when no test case can recognize the mutant and the first program. On the off chance that the mutant survives, then the test data is viewed as inadequate to investigate the fault. All things considered, the test data is reached out until such a mutant is killed. At some point, it is unrealistic to discover attest case that recognizes the yield of the mutant and that of the first program in which case the mutant is called identical. A sample syntactic mutation is shown in Algorithm 1.

Algoritm 1; A simple syntactic mutation:

```
Example: Let us consider the program M
        if (z = = x+y)
                do this ()
          else do that ()
Some of the Possible Mutants of M would be
    M1: if (z = =x-y)
         do this()
       else do that ()
       M2: if (z = >x*y)
          do this ()
       else do that ()
       M3: if (z>x/y)
          do this ()
       else do that ()
       M4: if (z >x+y)
          do this ()
       else do that ()
       M5: if (z <x+y)
             do this ()
       else do that ()
```

From Algorithm 1 in the event that the estimation of x = 2 and y = 2 then M2 is a correspondent mutant of M on the grounds that no conceivable test can ever kill this mutant. Mutation testing is ordinarily computationally costly in light of the fact that a program might have an extensive number of faults and there might be an expansive number of mutants for even a small software unit. In this manner, we have to produce test case in a manner that the test data make the execution of the program to achieve each mutated statement. There are a few methods were produced to diminish the computational expense one of them is selective mutation (Mresa and Bottaci, 1999). In specific mutation, the quantity of mutant is extensively diminished by dismissing low execution mutation operators. Along these lines, it lessens computational expense. The formulation of problem for the proposed methodology is given in this study.

Formulation of problem: Given an arrangement of '"n" test cases that should be handled on "m" test paths/sequences, the proposed method finds a subset of test cases that covers the given arrangement of test paths/sequences considering the priority limitations which boosts the mutation score and path coverage. Let, J = {1, 2, … , n} be the set of test cases to be used and M = {1, 2, … , m} be the set of test sequences. Let MS_j denotes the mutation score of test case. The mutation score of each test case is signified by a vector <MS_1, MS_2, MS_3, … , MS_n> and $r_{j,m}$ = 1, if test case 'j' is appropriate for test sequence 'm' and $r_{j,m}$ = 0, otherwise. The goal is to create quality test cases that can expose the number of errors as could be expected from the software under test within the low cost and less time. According to the test adequacy criteria, the mutation score and path coverage should be maximized for every test case during the process of test case generation and the size of the last test case set ought to be minimized. The mathematical model is given by:

$$\text{Max} MS_n(F_n) \quad (1)$$

$$Pcov_n(F_n) \quad (2)$$

Where:

F_n = Test case set name
MS_n = Mutation Score of test case n
$Pcov_n$ = Path coverage of test case n

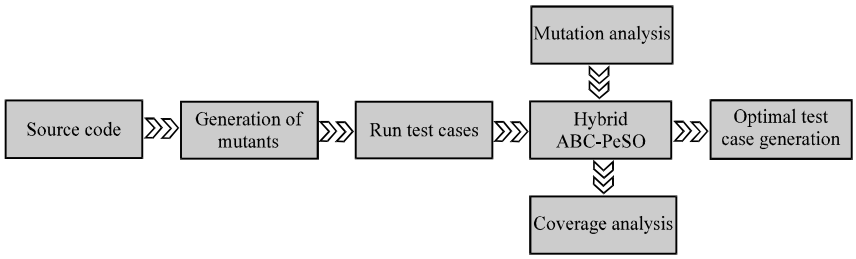

Fig. 2: Test case optimization framework

$$\text{Min Size}(J) \tag{3}$$

Subjected to:

$$MS_k \geq MS_{k-1} \quad k = 1 \text{ to } n \tag{4}$$

$$\sum r_{j,m} \leq 1,\ m>0 \text{ belongs to 'M' and 'j' belongs to 'J'} \tag{5}$$

$$MS_j \geq 0, j = 1 \text{ to } n \text{ and belongs to 'J'} \tag{6}$$

$$Pcov_n \geq 0, \text{'n' is test cas in the test casset} \tag{7}$$

$$\text{Mutation Score (MS)} = \frac{(\text{Dead mutants})}{(\text{Total mutants} - \text{Equivalent mutants})} \times 100 \tag{8}$$

$$\text{Path Coverage(PCov)} = \frac{(\text{Number of paths covered})}{(\text{Total Number of paths})} \times 100 \tag{9}$$

Where:

Mutants	=	Reformed program code
Dead mutants	=	Mutants which are killed by test case
Equivalent mutants	=	Mutants that cannot be killed by any of the test cases

The objective functions which are given in Eq. 1 and 2 maximizes the mutation score and path coverage of test case 'n'. The objective function in Eq. 3 minimizes the measure of the test case set. Limitation (Eq. 4) concentrates on the examination of mutation score of test cases and the limitation (Eq. 5) forces the priority connection between test cases and different limitations. It shows one test sequence can prepare one test case at any point of time. Constraint (Eq. 6) strengths mutation score to be non-negative. The limitation (Eq. 7) demonstrates the quality that path coverage metric takes taking into account the coverage of test case. The computation of mutation score and path coverage are given in the Eq. 8 and 9. Based on these objectives, the proposed test case optimization is done and the detailed representation is given in next study.

Framework of proposed test case optimization: In the proposed method, a novel hybrid algorithm namely, Hybrid Artificial Bee Colony-Penguin Search Optimization (ABC-PeSO) algorithm is designed for the process of test case optimization. The proposed method concentrates on generation of test cases and object oriented application optimization in which each class under test is the unit to be tested. The method is mainly applied to Java environment, class under test for each method is tested and it is repeated for different execution conditions. It is to be noted that in the classes of testing as units, the subsequent things are need to be performed:

- Variables to invoke the constructor
- Some of the methods that alter the state of the test object
- Method under test

If some of these variables are objects, it should be formed and put into a proper state. Hence, unit testing of test case of a class comprises of object formations of sequence (test object or variables), invocation of methods (Objects to be brought to proper state) and concluding invocation of test method. To provide a clear representation, let us consider an example, classes 'X' with method 'a' and a class 'Y' with method 'b', thus, the probable test cases are:

$$Xx = \text{new } X(); Yy = \text{new} Y(); y \times b(20); x \times a(45, y)$$

Adopt that the benchmark program triangle problem is the software under test which is provided as a distinct class to be tested which has methods. Now, the test case must contain all probable groupings of accessing these methods with different variables. However, this comprises a comprehensive enumeration of test cases which is not possible in reality. Therefore, the necessity is to create a test cases with basic set which improve the test case quality during iterations, so that they can fulfill the conditions provided by clients and at same time, the whole software is exercised and discloses as much possible number of errors. In this proposed methodology, a framework of optimization of test cases is been formed as shown in Fig. 2, using hybrid ABC-PeSO.

The need for hybrid algorithm for test case optimization is that, it is identified that in existing methods both simple optimization as well as population based search heuristics have only less consideration on fault detection capability of test cases in test case optimization problem. Here, we used two optimization algorithm called ABC and PeSO algorithm. In common, the ABC algorithm is performed in three phases:

- Employed bee phase: employed bee phase is to perform exploitation of food source
- Onlooker bee phase: onlooker bees are waiting in the hive for making decision
- Scouts bee phase: scout bees balance the exploration and exploitation capability of the algorithm

For the scout bees, it is divided into two parts. The scout bees in one part perform randomly search in the predefined region while in another part each scout bee select one non-dominated solution. In this proposed research, the random searches of both parts which are encapsulated in scout bee phase is employed by PeSO. Hence employed bee phase, onlooker bee phase and Scout bee phase composed of PeSO are the three phases, we comprised in the proposed research.

Artificial bee colony algorithm: An innovative swarm intelligence based optimizer is the Artificial Bee Colony (ABC) algorithm and it mimics the obliging foraging actions of a swarm of honey bees. ABC is used here for optimizing multi-modal and multi-variable continuous functions. Particularly, the number of control parameter in ABC is less compared with other population-based algorithms, thus make it easier to be implement. In the meantime, the performance of ABC is analogous and sometimes to the state-of-the-art meta-heuristics it is larger. Therefore, much interest has been paid and successfully applied to resolved inverse types of optimization issues. 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. 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 a bused 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".

The processes in scout bee are done by utilizing Penguin Search Optimization (PeSO) algorithm. Which facilitate the work of the scout bee phase more robust. In the proposed algorithm, a food source corresponds to a possible solution to the optimization problem and to the fitness of the associated solution the nectar amount of a food source is corresponded. 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.

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

$$F_i = \{F_{i,1}, F_{i,2}, \ldots, F_{i,d}\} \quad (10)$$

F_i represent the food source which is obtained by:

$$F_{i,d} = F_d^{mim}(1-r)+rf_d^{max} \quad (11)$$

where, r is a uniform random number in the range [0, 1] and F_d^{mim} F_d^{mix} are the lower and upper bounds for dimension d, respectively d = 1, ... , D.

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

$$S_{i,d} = r[(1+F_{i,d})-F_{k,d}] \quad (12)$$

Where:
d = Arbitrarily chosen from {1, ... , SN}such that $k \neq i$
r' = 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.

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 can be calculated by the factor of food source which is chosen by onlooker bee and the computation is expressed as follows:

$$pb_i = \frac{f_i}{\sum_{i=1}^{SN} f_i} \quad (13)$$

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

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 Eq. 11 the scout produces a food source.

Penguins search optimization algorithm: In this proposed methodology, we used a new meta-heuristic, called Penguins Search Optimization (PeSO) algorithm hybridization with ABC algorithm on basis of hunting behavior of penguins. The hunting procedure of penguins is more than fascinating 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 affixed number of dives, the penguins back on the ice to impart to its affiliate's depth (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.

Algorithm 2; Pseudo code of the algorithm PeSO:

```
Generate random population of P solutions (penguinsin) groups
Initialize the probability of existence of fish in the holes and levels
For i =1 to number of generations
For each individual i ∈ P do
While oxygen reserves are not depleted do
    Take random step
    Improve the penguin position using equation
    Update quantities of fish eaten for this penguin
End
End
    Update quantities of fish eaten for this penguins
    Redistributes the probabilities of penguins in holes and levels (these
    probabilities are calculated based on 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 fixed in 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:

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

where, rand () is a distribution random number and we have three solutions as best local solution, last solution and new solution. The computations in update solution (Eq. 14) are reiterated for each penguin in each group and 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. In this proposed method, for designing a tester we hybrid both Artificial Bee Colony (ABC) and Penguin Search Optimization (PeSO) the design of hybrid tester is show in next subsection.

Hybrid Artificial Bbee Colony-Penguin Search Optimization (ABC-PESO): In this research, we propose an optimized test suite produced by the tester which comprise of all possible statements and faults in program. In ABC algorithm on scout bee phase, if the quality of a solution cannot be enhanced after a predetermined number (limit) of trials, the food source is assumed to be abandoned and the equivalent employed bee becomes a scout. The scout will then produce a food source randomly and in order to improve the optimization performance of Scout bee phase, Penguin Search Optimization (PeSO) is utilized here. The pseudo code for test case optimization utilizing hybrid ABC-PeSO is given below. To implement any algorithm, the algorithm needs to be converted into pseudo code before programmatically emerging into an application. The pseudo code for the ABC-PeSO procedure is initiated with a collection of random particles (solutions), N. The ith particle is denoted by its position as a point in S-dimensional space where S denotes the number of variables.

Algorithm 3; Pseudo code of ABC-PeSO algorithm:

Step 1:In the initialization phase, the food sites (solutions) population is initialized by artificial scout bees and control parameters
Step 2: Search for an achievable state and assess the test node
Step 3: Initialize the present traversal path as cycle = 1
Step 4: Repeat
Step 5: Produce new test cases v_{ij} in the locality of $F_{i,d}$ for the search agent utilizing the beneath equation

$$F_{i,d} = F_d^{min}(1-r) + rf_d^{max}$$

where $F_{i,d}$ is the primary test case, F_d^{min} is the lowest number of test cases, F_d^{max} is the maximum number of test cases, r is a uniform random number in the range [0,1]

Step 6: Generate the test data
Step 7: Apply greedy selection procedure for the generated test data
Step 8: Traverse the software which under test with the test data generated and calculate the fitness value utilizing Eq. 1 and 2
Step 9: The onlooker chooses the test cases with the utmost fitness value and abandoned the rest
Step 10: This procedure is passed out till a specific test data with 100% fitness value and 0% fitness value is created which is given by

$$pb_i = \frac{fi}{\sum_{i=1}^{SN} f_i}$$

where pb_i is the probability function which indicates the probability with which the ith test data traversers an independent test path successfully
Step 11: Add the test case to the test database
Step 12: In the next iteration scout bees generate the new test data in our proposed method the scout bee phase is done by PeSO algorithm
Step 13: Initialize the population P and parameter initialization

- Gene: Number of generation (New Test Data)
- RO_2: Oxygen Reserves
- Best Sol: Best Solution

Step 14: Calculate best individual of population BGp
Step 15: While (iteration>gene) do
Step 16: For each individual do
Step 17: While (RO2>0) do
Step 18: Calculate the new solutions D_i utilizing Eq. 14
Step 19: Select the finest solutions of D_i
Step 20: End while
Step 21: End for
Step 22: If ($f(D_{id})$<f(BGp)) then
Step 23: BGp = D_{id}
Step 24: End If
Step 25: If (f(BGp)<f(BSol)) then
Step 26: BSol = BGp
Step 27: End if
Step 28: End while
Step 29: Retain the best solutions
Step 30: The best food source is memorized which are found so far
Step 31: Repeat step 5 to step 30 until stopping criteria meet

The flow chart of the proposed hybrid ABC-PeSO is shown in Fig. 3. From the pseudo code and flowchart, the explanation of the proposed method can be clearly understood and the implementation of the proposed method and the results with comparison and discussion are given in next study.

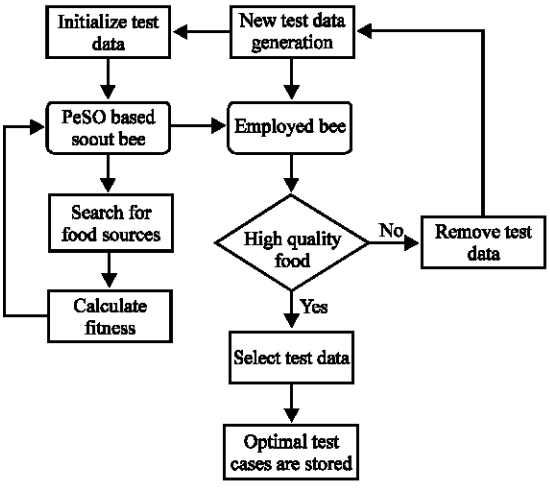

Fig. 3: Flowchart of hybrid ABC-PeSO

RESULTS AND DISCUSSION

In this study, we provide the results of preliminary experimental study carried out to assess the performance of the proposed method for automatic test data generation for mutation testing using hybrid ABC-PeSO. The experimental set up used for the proposed methodology, performance evaluation, performance comparison of the proposed methodology with existing works and the discussion are given in this section in detail.

Experimental set up: The proposed methodology implemented using the language of Java of Eclipse, Version 4.3 and using Intel i5 under a personal computer with 2.99 GHz CPU, 8 GB RAM and Windows 8 system. In our proposed method, we have used two benchmark programs as test beds one is triangle program and other one is next date 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 Next Date 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 http://web. soccerlab. polymtl.ca/repos/soccerlab/testing-resources/mutation testing/. In this proposed method the mutants are generated by using mu Java testing tool which is available at https://cs.gmu.edu/~offutt/mujava/. As triangle and next date does not reveal object oriented features, mutation was performed through μJava traditional operators; 94 and 104 min were created. The Triangle program is shown in Algorithm 4 and the next date program.

Algorithm 4; triangle program:

```
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[arge.length]
for9int i = 0 ; i<arge.length; i++)
{
s[i] = new java.lang.Float(args[i])
}
System.out.println9get Type(s))
}
int ret = 0
```

```
flat side1 = sides[0]
flat side2 = sides[1]
flat side3 = sides[2]
if (sides.length! = 3) {
ret = ILLEGAL ARGUMENTS
} else {
int triang = 0
if (side1 = = side2) {
triang = tring+1
if (side2 = = side3) {
triang = tring+2
}
if (side1 = = side3) {
triang = tring+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 and side1+side2<side3) {
ret = ISOCELES
} else {
if (triang = = 2 and side2+side3< side1) {
ret = ISOCELES
} else {
if (triang = = 3 and side1+side side3>
side2) {
ret = ISOCELES
} else {
ret = ILLEGAL
}
}
}
}
}
}
}
return ret
}
}
```

Algorithm 5; Next Date program:

```
Package Next Date
Public class next date
{
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 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) and (year
% 100 ! = 0)) || (year % 400 = = 0))
days in month = 28
break
}
default
days in month = 30
}
if (day <1 || day >days in month)
{
day = 1
if (month ! = 12)
{
month++
}
else
{
month = 1
year++
}

}
else
{
day++
}
message = day+"/"+month+"/"+year
}
}
}
System.out println(message)
}
}
```

In the proposed method, we have designed a novel ABC-PeSO algorithm for test case optimization here we have used hybrid technique, previously there are so many optimization algorithm are used for test case optimization like Genetic Algorithm (GA), Artificial Bee Colony (ABC) and so many algorithms. Since, a new hybrid technique is introduced there is need to prove that our proposed ABC-PeSO algorithm is better compared with other existing algorithms. Here a detail comparison is made with

four types of algorithms like Artificial Bee Colony (ABC), Penguin Search Optimization (PeSO), Particle Swarm Optimization (PSO) and Genetic Algorithm (GA).

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 (Masud *et al*., 2005). GA have a place with the bigger class of E volutionary 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 met heuristic search technique 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 represents to 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 a generation are chosen by objectives with a specific end goal to perform generation, i.e., crossover and/or mutation. At that point, new generation 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.

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 by Kennedy and Eberhart (1995). 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 fish schooling. 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 the best previous position of the entire population is called gbest. The result obtained by using PSO is shown in next study.

Performance evaluation and comparison: In this proposed method, the performance evaluation for the two programs triangle and next date which is analyzed based on mutation score and path coverage for the generated test report. The mutation score of individuals generated during various generation using proposed ABC-PeSO, ABC, PeSO, PSO and GA for triangle program is shown in Fig. 4 and the path coverage of test cases is shown in Fig. 5.

From Fig. 4, it can see that mutation score obtained by the different method our proposed method has achieved highest mutation score of 95% whereas the mutation score of PeSO, PSO, ABC and GA are 88, 70, 72 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.

From Fig. 5, it can be noted that the proposed method has achieved maximum path coverage with 96% to that of path coverage achieved by 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 proposed ABC-PeSO, ABC, PeSO, PSO and GA for Next Date program is shown in Fig. 6 and 7.

Similarly, for the case of the next date program, the mutation score obtained by the proposed method has achieved high value of 98% to that of mutation score obtained by PeSO, PSO, ABC and GA are 85, 71, 76 and 60% for the particular generation.

Likewise for the case of path coverage also, the proposed method has attained a high value of 98% to that of the path coverage of PeSO, PSO, ABC and GA are 86, 79, 80 and 55% for the particular generation. This result shows that the proposed hybrid ABC-PeSO has achieved better results. So, far from the results obtained from two benchmark programs we can see that our proposed method that is hybrid ABC-PeSO has attained high mutation score of 95 and 98% for triangle and nextdate program also it attained path coverage of 96 and 98% for triangle and next date program this shows the performance as well as the importance of the proposed method in the field of software testing. Our proposed method has achieved better results than previously used algorithms like ga, pso as well as abc which can be clearly seen from the results presented. Finally, from the results

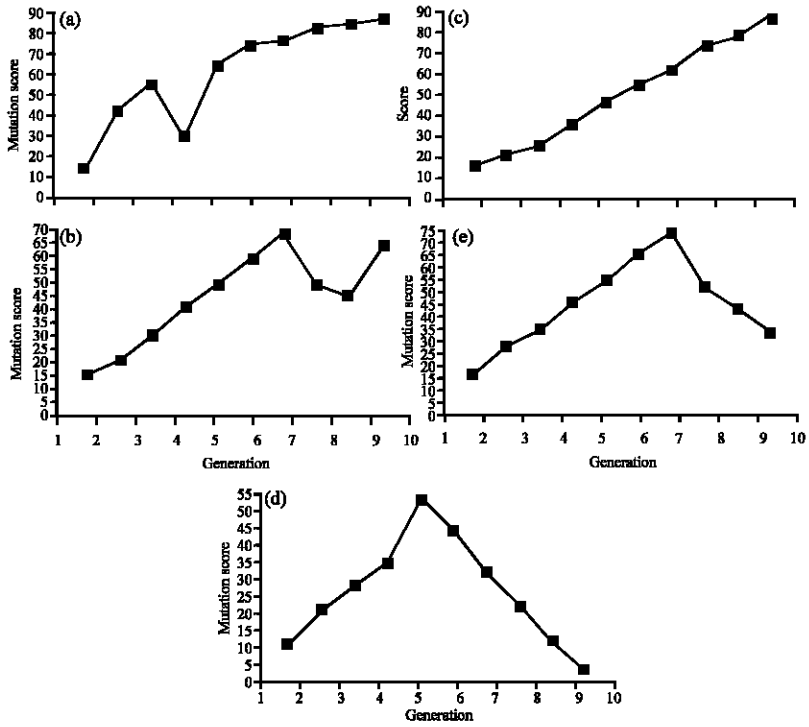

Fig. 4: Mutation score of triangle program: a) Mutation score for ABC-PeSO; b) for PeSO; c) for PSO; d) for ABC and e) for GA

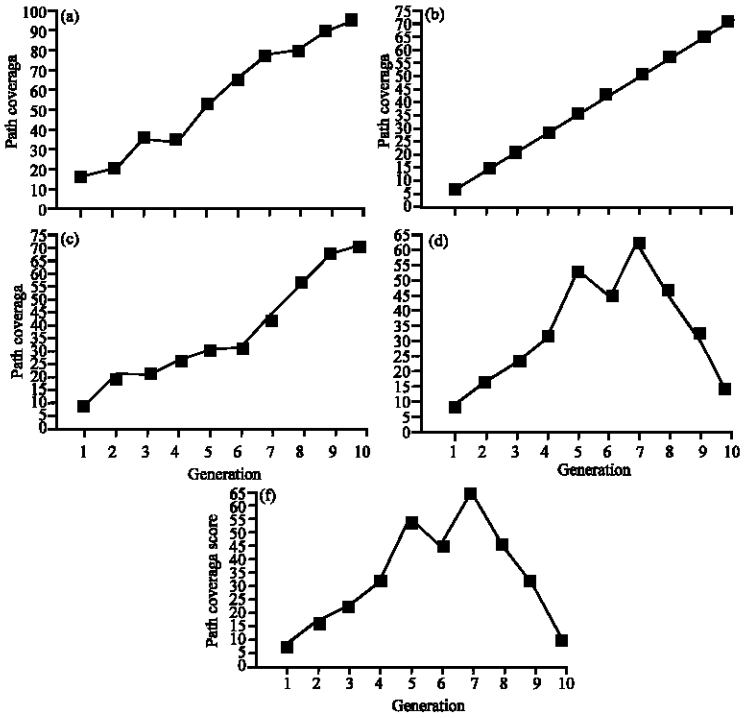

Fig. 5: Path coverage of triangle program: a) Path coverage for ABC-PeSO; b) for PeSO; c) forPSO; d) for ABC

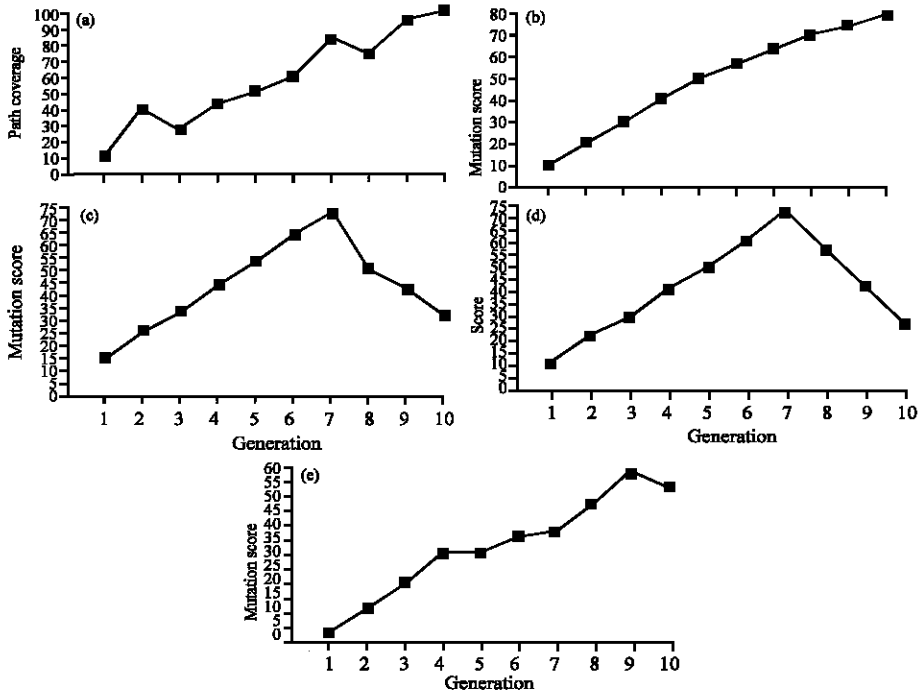

Fig. 6: Mutation score of next date program: a) Mutation score for ABC-PeSO; b) F for PeSO; c) For PSO; d) For ABC and e) For GA

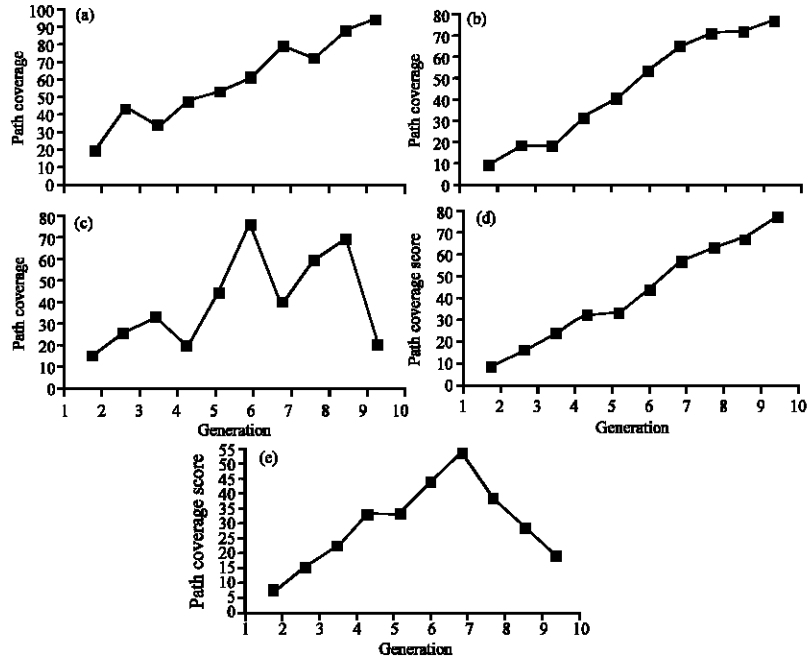

Fig. 7: Path coverage of next date program: a) Path coverage for ABC-PeSO; b) For PeSO; c) For PSO; d) For ABC and e) For GV

we can conclude that our proposed method has generated quality a test case for which clears more errors in the program with less cost. The quality of the test cases can be easily analyzed from the mutation as well as path coverage in both these we achieved maximum results.

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. In this proposed method, we have proposed a novel hybrid ABC-PeSO to decrease the test data generation cost and time in the context of mutation testing. The proposed method is implemented on Java working platform and tested on two benchmark programs they are triangle and next date. Experimental results obtained on two programs showed that the proposed hybrid ABC-PeSO performed well and produces satisfactory results better than other algorithms like ABC, PeSO, PSO and GA this shows the importance of this newly designed method in the field of software testing.

REFERENCES

Ahmed, B.S. and K.Z. Zamli and C.P. Lim, 2012. Application of particle swarm optimization to uniform and variable strength covering array construction. Applied. Soft Comput. J., 12: 1330-1347.

Ahmed, B.S. and K.Z. Zamli, 2011. A variable strength interaction test suites generation strategy using particle swarm optimization. J. Syst. Software, 84: 2171-2185.

Ahmed, B.S., M.A. Sahib and M.Y. Potrus, 2014. Generating combinatorial test cases using Simplified Swarm Optimization (SSO) algorithm for automated GUI functional testing. Eng. Sci. Technol. Int. J., 17: 218-226.

Anand, S., E.K. Burke, T.Y. Chen, J. Clark and M.B. Cohen *et al*., 2013. An orchestrated survey of methodologies for automated software test case generation. J. Syst. Software, 86: 1978-2001.

Baker, R. and I. Habli, 2013. An empirical evaluation of mutation testing for improving the test quality of safety-critical software. IEEE. Trans. Software Eng., 39: 787-805.

Bansal, A., 2014. A comparative study of software testing techniques. Intl. J. Comput. Sci. Mob. Comput., 3: 579-584.

Belli, F., C.J. Budnik, A. Hollmann, T. Tuglular and W.E. Wong, 2016. Model-based mutation testing-approach and case studies. Sci. Comput. Program., 120: 25-48.

Bryce, R.C., S. Sampath, J.B. Pedersen and S. Manchester, 2011. Test suite prioritization by cost-based combinatorial interaction coverage. Intl. J. Syst. Assur. Eng. Manage., 2: 126-134.

Debroy, V. and W.E. Wong, 2014. Combining mutation and fault localization for automated program debugging. J. Syst. Software, 90: 45-60.

Fraser, G. and A. Arcuri, 2015. Achieving scalable mutation-based generation of whole test suites. Empirical Software Eng., 20: 783-812.

Fraser, G. and A. Zeller, 2012. Mutation-driven generation of unit tests and oracles. IEEE. Trans. Software Eng., 38: 278-292.

Garvin, B.J., M.B. Cohen and M.B. Dwyer, 2011. Evaluating improvements to a meta-heuristic search for constrained interaction testing. Empirical Software Eng., 16: 61-102.

Habibi, E. and S.H. Mirian-Hosseinabadi, 2015. Event-driven web application testing based on model-based mutation testing. Inf. Software Technol., 67: 159-179.

Hamlet, R.G., 1977. Testing programs with the aid of a compiler. IEEE Trans. Software Eng., SE-3: 279-290.

Jia, Y. and M. Harman, 2009. Higher order mutation testing. Inf. Software Technol., 51: 1379-1393.

Jia, Y. and M. Harman, 2011. An analysis and survey of the development of mutation testing. IEEE. Trans. Software Eng., 37: 649-678.

Kennedy, J. and R. Eberhart, 1995. Particle swarm optimization. Proc. IEEE. Intl. Conf. Neural Netw., 1: 1942-1948.

Kuliamin, V.V. and A.A. Petukhov, 2011. A survey of methods for constructing covering arrays. Program. Comput. Software, 37: 121-146.

Mao, C., X. Yu, J. Chen and J. Chen, 2012. Generating test data for structural testing based on ant colony optimization. Proceedings of the 12th International Conference on Quality Software (QSIC'12), August 27-29, 2012, IEEE washington, USA. isBN:978-0-7695-4833-3, pp: 98-101.

Masud, M., A. Nayak, M. Zaman and N. Bansal, 2005. A strategy for mutation testing using genetic algorithms. Proceedings of the Canadian Conference on Electrical and Computer Engineering, May 1-4, 2005, Saskatoon, Sask, pp: 1049-1052.

Mresa, E.S. and L. Bottaci, 1999. Efficiency of mutation operators and selective mutation strategies: An empirical study. Software Test. Verif. Reliab., 9: 205-232.

Nayak, N. and D.P. Mohapatra, 2010. Automatic Test Data Generation for Data Flow Testing using Particle Swarm Optimization. In: Contemporary Computing, Ranka, S. (Ed.). Springer, Berlin, Germany isBN:978-3-642-14824-8, pp: 1-12.

Nidhra, S. and J. Dondeti, 2012. Blackbox and whitebox testing techniques: A literature review. Intl. J. Embedded Syst. Appl., 2: 29-50.

Nie, C. and H. Leung, 2011. A survey of combinatorial testing. ACM. Comput. Surv., 43: 11-29.

Nie, C., H. Wu, X. Niu, F.C. Kuo and H. Leung *et al*., 2015. Combinatorial testing, random testing and adaptive random testing for detecting interaction triggered failures. Inf. Software Technol., 62: 198-213.

Pachauri, A. and G. Srivastava, 2013. Automated test data generation for branch testing using genetic algorithm: An improved approach using branch ordering, memory and elitism. J. Syst. Software, 86: 1191-1208.

Torres-Jimenez, J. and E. Rodriguez-Tello, 2012. New bounds for binary covering arrays using simulated annealing. Inf. Sci., 185: 137-152.

Usaola, M.P. and P.R. Mateo, 2010. Mutation testing cost reduction techniques: A survey. IEEE. Software, 27: 80-86.

Yuan, X., M.B. Cohen and A.M. Memon, 2011. GUI interaction testing: Incorporating event context. IEEE Trans. Software Eng., 37: 559-574.