International Journal of Control and Automation
Vol. 11, No. 4 (2018), pp.45-62
http//dx.doi.org/10.14257/ijca.2018.11.4.05

Particle Swarm Optimization based Optimal Directed Random Testing for Reducing Interactive Faults

K. Koteswara Rao

Professor, CSE Department, VITAM & Research Scholar @JNTUK
Visakhapatnam, AP, India
koteswara2003@yahoo.co.in

Abstract

The expectation of programming testing is to finding the blunders in programming. Programming testing is the way toward approving and confirming that a program capacities accurately. Irregular testing produces test inputs haphazardly from the info space of the product under test. To produce arbitrary experiments every time, which will contain some closeness? With a specific end goal to conquer these issues, we will propose a procedure for decreasing the shortcomings in view of ideal experiments produced from coordinated arbitrary testing. In proposed strategy, we will produce a proficient irregular testing experiment in light of the question conduct reliance display. In this examination, the ideal data sources will be produced in light of Particle Swarm Optimization (PSO) which will lessen the unlawful information sources and proportionate sources of info. To diminish the blame inclination, PSO utilizes the scope measurements of the experiments. Our proposed strategy will prunes the information space by consolidating the past contribution with the present one furthermore increment adaptability and viability in the period of programming testing.

Keywords: *-software testing, test case, object behavior dependence model, Particle Swarm Optimization, coverage metrics*

1. Introduction

Arbitrary testing is a discovery programming testing method where projects are tried by creating irregular, free information sources. Among the different programming testing strategies, Random Testing (RT) is the most basic procedure. It is basic in idea. It's regularly simple to apply, can frequently practice the product under test in surprising ways, and has shown adequacy in recognizing disappointments [1]. Arbitrary number is generally utilized as a part of cryptographic applications, which is for the most part utilized as key. Since the security of key thoroughly relies on upon the sum and arbitrariness of itself, it's imperative to deliver irregular numbers for cryptographic applications [7]. Irregular number generators broadly utilized in business applications don't entirely ensure these necessities [6]. Notwithstanding, a noteworthy test for these methodologies is the undetermined, two-dimensional, and combinatorial colossal information space that they need to investigate and practice naturally [9].

The time test information era utilizing half breed technique that takes the benefits of both static and element strategy was done [14]. Programming testing remains a to a great degree exorbitant movement in the product designing lifecycle, and in that capacity, its robotization keeps on being of high concern [4]. To produce tests that cover the majority of the branches in a class, the class must be instantiated, and maybe a technique call arrangement ought to be created to put the question into a specific state [5]. Creating unit test suites naturally is a critical commitment towards enhancing programming quality, and

Received (December 11, 2017), Review Result (February 23, 2018), Accepted (March 5, 2018)

ISSN: 2005-4297 IJCA

procedures like inquiry based programming testing dynamic typical execution can productively deliver test suites accomplishing high code scope [3]. Model-based framework testing of utilizations with a GUI front-end to be more financially savvy and proficient contrasted with their conventional record-then-replay partners [8]. A change of RT exists to enhance its proficiency, for example, Adaptive Random Testing (ART) ART used to test numerical projects, in light of disappointment examples which comprises of three classes: square example, strip example, and point design [10]. Be that as it may, ART is less effective than irregular testing due to the additional errand of guaranteeing notwithstanding spreading of experiments, where the proficiency is measured as far as an ideal opportunity to produce an experiment [13]. Random test information era, creates test information in light of specific arbitrary sources of info frame some dissemination. Way situated and basic methodologies utilize the program's control stream diagram for test information era; they select a way, and utilize a procedure, for example, typical execution for era of test information. Objective arranged test-information era approaches select contributions to execute the chose objective, for example, explanation, condition scope, choice scope, regardless of the way taken [12]. The reason for a finding is a test suite, and regularly the current test suite is not upgraded for creating amazing symptomatic reports. Henceforth, is critical to produce tests to enhance finding. There are numerous accessible test era methods Search-Based Software Testing (SBST). Worldwide inquiry calculations are Genetic Algorithms [15].

2. Related Works

Bo Yu, and Zeliang Pang [16] have proposed the enhanced uniform plan procedure to outline test information for planning test set was exhibited. Tests demonstrate that that the enhanced uniform plans procedure was steadiness and could supplant the mix test strategy right now for the test time being restricted in view of the speculation testing.

Padgham *et al.*, [17] have proposed a model-based prophet era strategy for unit testing conviction seek goal specialists. They build up a blame model in view of the elements of the center units to catch the sorts of issues that might be experienced and characterize how to consequently produce an incomplete, uninvolved prophet from the specialist plan models Line both the blame model and the prophet era by testing 14 operator frameworks. More than 400 issues were raised, and these were examined to find out whether they spoke to bona fide blames or were false positives. They found that more than 70 percent of issues raised were demonstrative of issues in either the plan or the code. Of the 19 checks performed by their prophet, flaws were found by everything except 5 of these checks. They additionally observed that 8 out the 11 blame sorts distinguished in their blame model displayed no less than one blame. The assessment demonstrates that the blame model was a gainful conceptualization of the issues not out of the ordinary in specialist unit testing and that the prophet could locate a considerable number of such blames with generally little overhead as far as false positives.

Bestoun S. Ahmed *et al.*, [18] have proposed a procedure to be utilized for GUI utilitarian testing. The system produces the required experiments for the GUI under test utilizing the combinatorial plan and after that it removers the undesirable experiments. The Simplified Swarm Optimization hypothesis was utilized to advance the experiments considering the combinatorial plan ideas. By producing the experiments the tests connected to a genuine contextual investigation to test the viability of the procedure. The key knowledge in that work was to influence the way that combinatorial outline could be utilized productively and viably for various testing contextual analyses and it could be utilized additionally as a part of GUI testing.

Andrea Arcuri and Lionel Briand [19] have proposed a few hypotheses depicting the likelihood of irregular testing to distinguish connection blames and contrast the outcomes with combinatorial testing when there are no requirements among the elements that can be

a piece of an item. Irregular testing turns out to be significantly more viable as the quantity of elements increments and focalizes toward equivalent viability with combinatorial testing. Given that combinatorial testing involves critical computational overhead within the sight of hundreds or a great many components, the outcomes recommend that there are sensible situations in which arbitrary testing may beat combinatorial testing in huge frameworks. Moreover, in like manner circumstances where test spending plans are obliged and dissimilar to combinatorial testing; arbitrary testing can even now give least certifications on the likelihood of blame identification at any connection level. At the point when limitations were available among components, then arbitrary testing can admission self-assertively more regrettable than combinatorial testing.

Leandro L. Minku *et al*., [20] have proposed a hypothetical examination has down to earth suggestions in view of it; they determine an enhanced EA plan. That incorporates normalizing representatives' commitment for various errands to guarantee they are not working additional time; a wellness capacity that requires less pre-characterized parameters and gives a reasonable slope towards plausible arrangements; and an enhanced representation and change administrator. Both that hypothetical and observational results demonstrate that our outline was extremely powerful. Joining the utilization of standardization to a populace gave the best results their investigations, and standardization was a key part for the handy viability of the new outline.

JunpengLv *et al.*, [21] have proposed a half and half approach that utilizations AT and arbitrary parcel testing (RPT) in a substituting way. The inspiration for that approach was that both systems were utilized to such an extent that the hidden computational many-sided quality of AT was diminished by bringing RPT into the testing procedure without influencing the deformity location adequacy. A contextual analysis with seven genuine subject projects was introduced. The trial comes about exhibit that novel methodology extensively diminishes the computational overhead of the first AT technique yet at the same time beats the unadulterated RT procedure and PT system as far as the quantity of experiments used to distinguish and expel a given number of imperfections.

3. Problem Definition

In this section briefly discussed the problem definition

- Object-Oriented Software Testing needs to manage new issues presented by the O-O components, for example, exemplification, legacy, polymorphism, and element authoritative.
- Automation of testing won't be that much beneficial as far as tedious and cost.
- Reducing the experiments at the season of creating dodge era of repetitive experiments while the other one can be viewed as a streamlining issue.
- Valid test arrangements are create from the state show. Their experiment era approach basically changes over the case outline issue into an arranging issue.
- Finding flaws competitor can be characterized in term of the generally known insignificant hitting set (MHS) issue.
- To distinguish infeasible ways for sparing computational time.
- The variable esteem in image might be an exceptionally complex to unravel.
- The number of ways is unbounded and choosing whether a way is practical or not is an un decidable issue all for all situation.
- The number of execution might be increasingly and it might come up short experiment regardless of the possibility that one exit at long last if there should arise an occurrence of infeasible way it won't end.
- The issue of infeasible way can be wiped out just by considering a design which has no way selector stage.

- The run time depends on characteristics of the problem instance in particular the problem

4. Proposed Method

Software testing is huge and different field, replicating the different requirements that software artifacts must satisfy the different activities involved in testing and the different levels at which software can be checked. Arbitrary testing generates test inputs arbitrarily from the input space of the software under test. The basic feature of a random test generation technique is that it produces test inputs at arbitrary from a grammar or some other formal artifact explaining the input space. The most important purpose of the suggested method is to increase an effective technique for decreasing interactive faults based on optimal test case in direct random testing. For the generation of test cases the suggested method is employing Object Behavior Dependence Model (OBDM). The resulting inputs are fed to the PSO; the specified process of executed method is illustrated in Figure 4.1. The block diagram of the suggested method is illustrated in below,

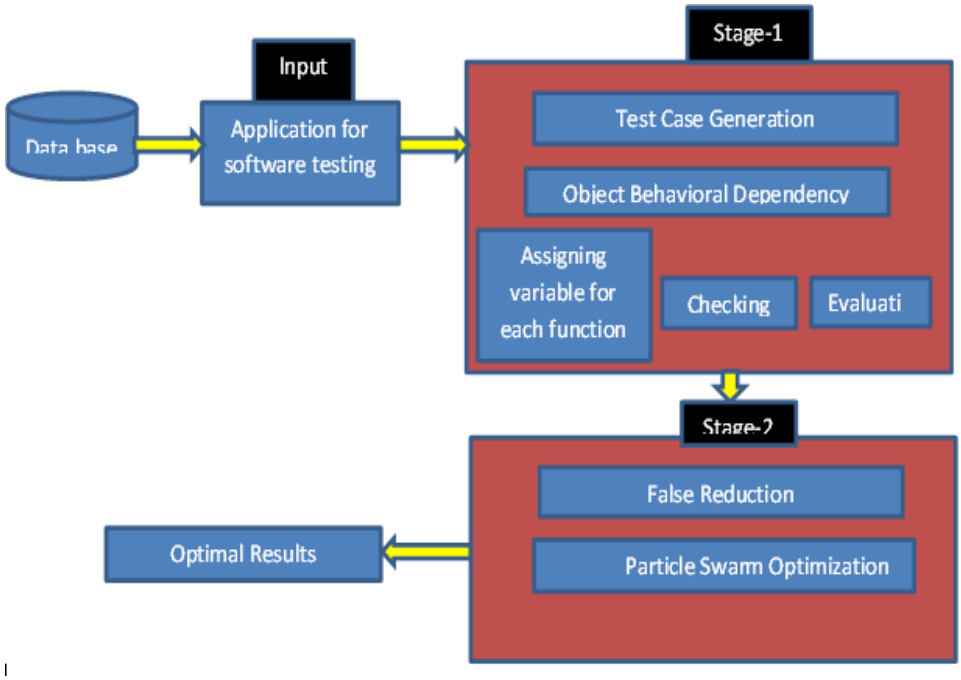


Figure 4.1. Block Diagram of Proposed Method

OBDM:In sequence diagram set of nodes representing objects (O_b) and set of edges that indicate the function (F). Where, $F \in S_f$ represents the synchronous function. Function has the following six attributes and has a direct dependency between the source and destination objects.

$F_{source} \in O_b$ - Source of the function

$F_{dest} \in O_b$ - Destination of the function and where $F_{source} \neq F_{dest}$

F_{name} - Name of the function

$F_{BW} \in S_f$- Backward navigable function and where, $F_{BW} \neq F$ it is denoted as "-".

F_{ER} - Probabilistic execution rate of a function in a Sequence Diagram and where, $0 \leq F_{ER} \leq 1$ and the default value is 1.F_{EER}- Expected execution rate of a function in a Sequence Diagram and where, $0 \leq F_{EER} \leq 1$ and the default value is 1.We consider a branch control structure of a source code, in which the execution rate of a function may be affected.

Consider a function is in an alt combined fragment and only when the condition in the fragment is satisfied. If the function is executed within this condition fragment, then the probability of execution rate of a function is 0.5. Otherwise, the default value is 1. The expected execution rate of a function is a probability of the execution rate of a sequence diagram. In other words, it is the probability of the execution time for the total number of functions in a particular class to the execution time for the total number of functions in the whole input application. The function in a sequence diagram is executed only when it is activated. The default value of F_{EER} is also 1. Our proposed method has two stages namely,

1. Test case generation
   - ❖ Object Behavior Dependence Model (OBDM)
2. False reduction
   - ❖ Particle Swarm Optimization (PSO)

Stage 1:

4.1. Test Case Generation

The suggested method produces the test cases based on the Object Behavior Dependence Model (OBDM). The application which we are checking, takes as an input for object behavior dependence model in software testing. Each application has the number of function that is employed for the generation of test case. The suggested OBDM method principally focuses on the functions and coverage metrics of the application that we are applied for the test case generation. This will avoid generating duplicate and insignificant test cases. The function name is symbolized as a variable in our executed method. The execution of the OBDM shown bellow

INPUT APPLICATION PATH : src\bank
TOTAL NUMBER OF CLASSES IN INPUT FILE :: 47

1 : src\bank\ABank1.java

2 : src\bank\Acounter.java
……
……
47 : src\bank\Withdrawal.java

FINDING FUNCTIONS OF EACH CLASSES
FUNCTION NAME AND CORRESPONDING CLASSES
main src\bank\ABank1.java
ABank1 src\bank\ABank1.java
……
……
wd src\bank\Withdrawal.java
TOTAL NUMBER OF FUNTIONS IN ALL CLASSES : 108

In flowchart (Figure 4.2) the overall process of test case generation is illustrated and specifically made cleared below with some examples,

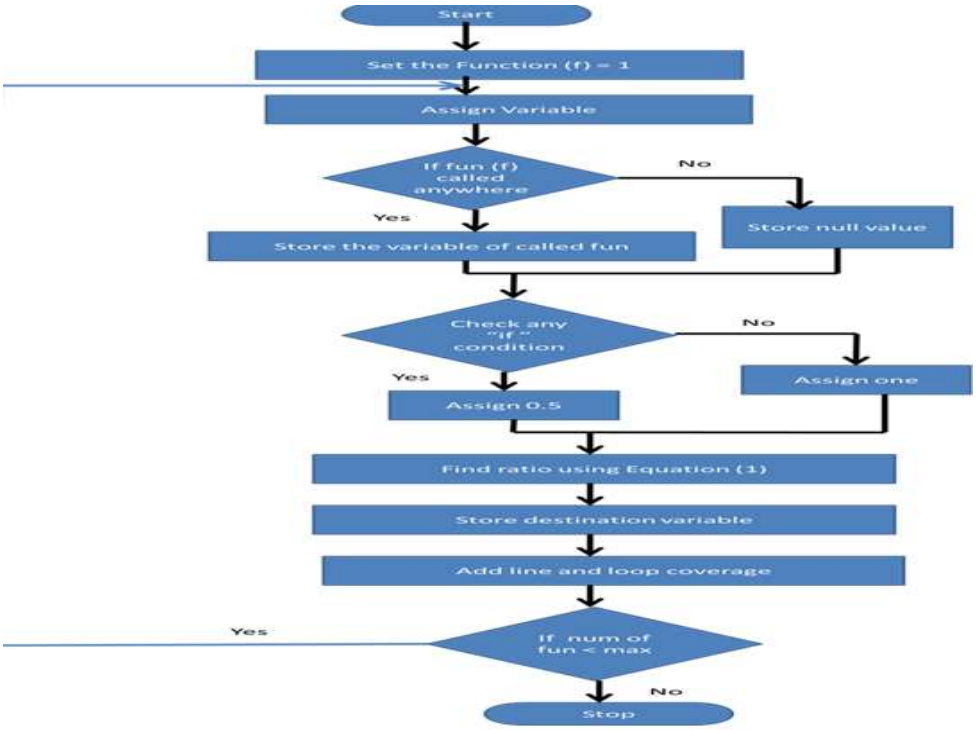


Figure 4.2. Flowchart for Test Case Generation using OBDM

Figure 4.2. illustrates the Generation process of test case. Reflect on each function as the source function. For each source function, a variable name is specified, and then each function is verified to find whether it is previously allocated for some other task. If it is previously been allocated, next the related variable name should be assigned in the test case, or else, allocate as illogical. If any function contains of "if" condition then allocates 0.5 values in the test case or else assign one. For the given function ratio will be worked out and at last symbolize the destination function. After that, attach all the test case to the coverage matrix. Here we are assuming the probability value, any function consist of “if” condition we assume 0.5 values otherwise one.

FINDING IF VALUE OF EACH FUNCTIONS. IF FUNCTIONS CONTAINS IF STATEMENT MEANS ASSIGN VALUE IS 0.5 OTHERWISE 1
VALUE AND ITS CORRESPONDING FUNCTION
1.0 :: main
1.0 :: ABank1
……..
0.5 :: wd
FINDING OBDM VALUE
O1 --> main(- ,1.0, 0.06481481481481481) --> O1
[O1, main, -, 1.0, 0.06481481481481481, O1, main]
…….
[O47, wd, -, 1.0, 0.018518518518518517, O35, menu]
SOURCE ID FUN NAME PARENT NAME IF VALUE RATIO VAL DES ID CLASS NAME
[O1, main, -, 1.0, 0.06481481481481481, O1, main]
……
……
[O47, wd, -, 1.0, 0.018518518518518517, O35, menu]

Ratio value will be calculated for the given function and finally represent the destination function. Then add all the test case to the coverage matrix.

$$Ratio = \frac{how\ much\ time\ call\ the\ other\ function}{Total\ \ function} \quad (1)$$

The suggested method employs only the line coverage and loop coverage from the coverage metrics.

FINDING RATIO VALUES. NUMBER OF TIMES THE FUNTION CALLED BY OTHER FUNCTIONS // TOTAL NUMBER OF FUNCTIONS
RATIO VALUE AND ITS FUNCTION NAME
0.06481481481481481 :: src\bank\ABank1.java
0.06481481481481481 :: src\bank\ABank1.java
……
0.009259259259259259 :: src\bank\Withdrawal.java

Line Coverage:

Line coverage is as well identified as the statement coverage or segment coverage. Only correct conditions are wrapped by line coverage. It as well measures the quality of the code and makes sure the flow of different path in that code.

$$line\ coverage = \frac{number\ of\ \ lines\ \ exercised}{total\ \ number\ of\ \ lines} \quad (2)$$

FINDING LINE COVERAGE
CLASS NAME FUNCTION NAME LINE COVERED BY IF PART STATEMENT LINE COVERED BY ELSE PART STATEMENT
[ABank1, main, 0, 0]
[ABank1, ABank1, 0, 0]
……
[Testing, errorread, 0, 0]

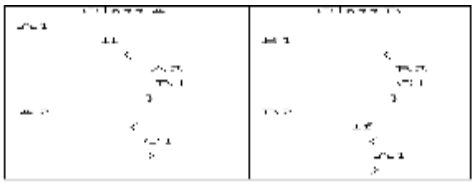
Loop coverage:

This coverage metrics reports whether each loop body is implemented zero times, precisely once and more than once. This metrics reports whether loop body is implemented precisely once and more than once for do-while loops. And as well, while-loops and for-loops perform more than once.

FINDING LOOP COVERAGE
CLASS NAME FUNCTION NAME LINES COVERED BY LOOPING STATEMENT TOTAL NUMBER OF LINES IN FUNCTION
[ABank1, main, 0, 5]
[ABank1, ABank1, 0, 41]
……
[Testing, errorread, 0, 13]
COMBINING LINE COVERAGE AND LOOP COVERAGE VALUES
[ABank1, main, 0, 0]

……
[Testing, errorread, 0, 0]

This data is not accounted by other coverage metrics. For example, reflect on one application; it has two classes with four functions here the function names are symbolized as a variable one. The specified process illustrated in table.



Variable name for the functions A1, A2, B1, B2 and C1 are F1, F2, F3, F4 and F5. In our proposed method, the test case contains source function name, probability value, ratio value, destination function name, line coverage and loop coverage. Test case generation process with the corresponding example is given below,

GENERATING TESTCASES BY COMBINING OBDM VALUES LINE COVERAGE AND LOOP COVERAGE VALUES
[O1, main, -, 1.0, 0.06481481481481481, O1, main, 0, 0]
[O1, main, -, 1.0, 0.018518518518518517, O5, main, 0, 0]
……
……
[O46, main, -, 1.0, 0.08333333333333333, O45, findrf, 0, 0]
TOTAL NUMBER OF TESTCASES :: 661

We are taking the input function is banking application. It contains nearly 47 classes and 108 functions. Total number of test case generation in stage 1 is around 661. Next resulting test case generation is fed to the PSO. Since we will produce test cases in each time it encloses some resemblance on each time. Suggested method employs the genetic algorithm in order to decrease the fault occurrence of the test cases.

Stage 2:

4.2. False Reduction

False reduction is basically defined as the reduction of added parameters which are not required for processing. In our proposed method, we generate enormous test cases in which some of those test cases are not required for processing. Also there may be similar test cases that are being generated at each time interval. In order to select the concerned test cases we require some efficient techniques. The next stage of the suggested method is false reduction by means of the PSO. Here in our proposed method genetic algorithm for false reduction will obtain optimized results. In this research, the optimal inputs will be produced based on Particle Swarm Optimization (PSO) which will decrease the illegal inputs and equivalent inputs. The overall procedure of PSO.

4.2.1. Particle Swarm Optimization: Particle swarm optimization is naturally motivated computational inquiry and improvement technique created in 1995 by Ebberhart and Kennedy on the social conduct of fowls running or fish tutoring. Fundamental PSO is proper for basic advancement issue .PSO calculation imitates from conduct of creature society that

1. Do not have pioneer
2. Will discover nourishment by irregular
3. Follow one of the individuals from the gathering that has the nearest position with sustenance source (potential arrangement)
4. Flocks will accomplish their best condition through correspondence among the individuals. Who as of now have better arrangement?
5. Animal which has a superior condition will illuminate to its rushes and other will move to that place

6. This process will be continued (happen over and over) until the best condition or nourishment source watched
7. PSO comprises of swarm of particles, while a molecule speak to a potential arrangement

Investigation is the capacity of hunt calculation to investigate distinctive districts of pursuit space keeping in mind the end goal to find a decent ideal The swarm fly through hyperspace has two vital thinking capacities

1. Their (Pb) memory of their own best position(local best)/claim position
2. Their neighborhoods position (worldwide best) (gb)

Position of the molecule is affected by speed Position of the particle is influenced by velocity

X_i^k denote current position of agent i at iteration k

The position of the particle is changed by adding velocity V_i^{k+1} to the current position

$$\mathbf{X_i^{k+1} = X_i^k + V_i^{k+1}} \quad (1)$$

$$\mathbf{V_i^{k+1} = V_i^k + c_1 r_1 (P_{besti} - X_i^k) + c_2 r_2 (g_{best} - X_i^k)} \quad (2)$$

With acceleration coefficients c_1 and c_2

Random vector r_1 and r_2

Procedure:

1. Assume the extent of gathering of molecule is N
2. Generate the underlying populace X with range X(B) and X(A) by irregular request to get X1, X2… … .. Xn
3. After that the molecule "k" and speed at emphasis "i" are meant as Xik, Vik consequently introductory process will be X1(0), X2(0)… … ... Xn(0), Xi(0) (i = 1,2,3… … ..n) is called molecule or vector directions of the molecule, (for example, chromosomes in hereditary calculation)
4. Evolution of target capacity esteem is communicated by f[X1(0)], f[X2(0)],… f[Xn(0)] At that point figure speed of all particles, all particles move towards the ideal point with a speed. At first all the molecule speed is thought to be zero, set emphasis k = 1 ,

At the k th emphasis discover two essential parameters for molecule "i"

a. The best value of $\mathbf{X_i^k}$ particle i at iteration k and declare P_{best} at k
b. Calculate velocity of particle i at iteration k using formulae(2) where c_1 and c_2 learning rates for individual and group, r_1 and r_2 are random numbers distributed in binary 0 and 1

   c_1, c_2 represents weight of memory position of particle towards memory position of group generally c_1 and c_2 are taken as 2

   Multiply c_1 r_1, c_2 r_2 ensure that particle will approach
c. Calculate the position of the particle 'i'

   $\mathbf{X_i^{k+1} = X_i^k + V_i^{k+1}}$
d. Find the fitness f[$X_1^{(k)}$], f[$X_2^{(k)}$],……. f[$X_n^{(k)}$]

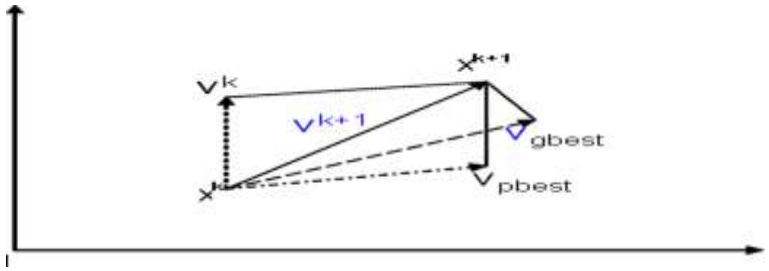


Figure 4.3. Concept of Modification of a Searching Point by PSO

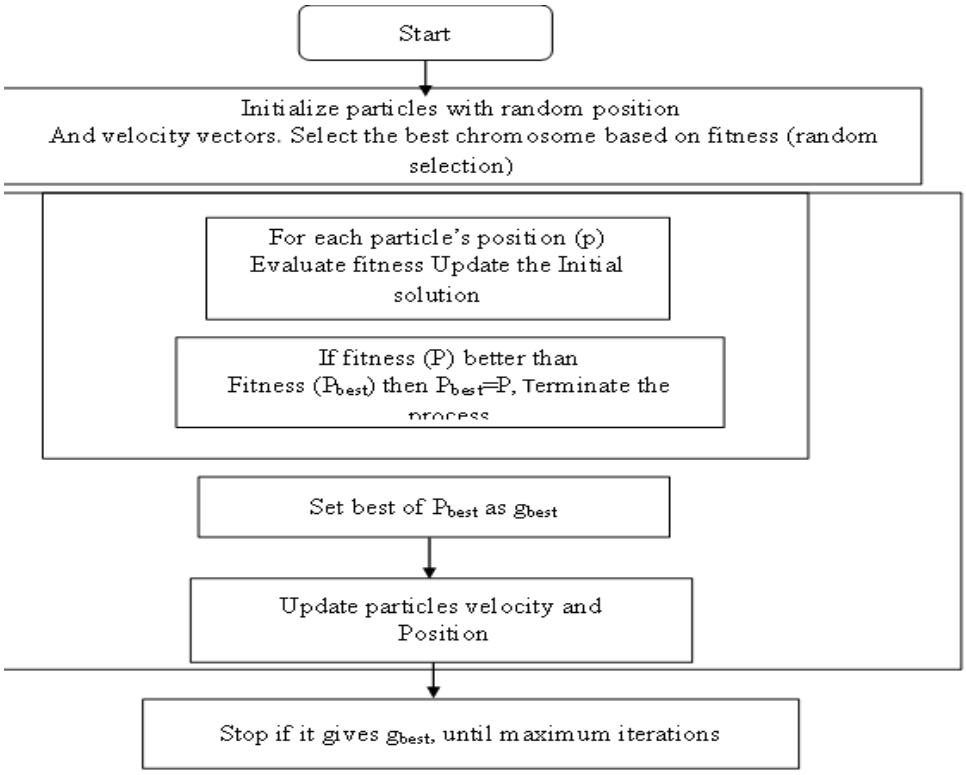


Figure 4.4. Flowchart for General PSO Algorithm

Step1: Initialization

Initialize particles with random position and velocity vectors
[O14, main, -, 1.0, 0.018518518518518517, O2, count, 0, 0] ini
[O13, AMtrans, -, 1.0, 0.018518518518518517, O35, menu, 4.0, 0.0594059405940594] ini
[O10, ALtrans, -, 1.0, 0.046296296296296294, O46, main, 0, 0] ini
[O37, Mtrans, -, 1.0, 0.018518518518518517, O35, menu, 1.0, 0.05405405405405406] ini
[O21, witdis, -, 1.0, 0.018518518518518517, O35, menu, 0, 0] ini
[O19, APlo, -, 1.0, 0.018518518518518517, O10, main, 0, 0] ini
[O42, Pplo, -, 1.0, 0.018518518518518517, O17, main, 1.25, 0.033707865168539325] ini
[O45, main, -, 0.5, 0.08333333333333333, O45, findprio, 0, 0] ini
[O12, Delet, -, 1.0, 0.06481481481481481, O1, ABank1, 1.5, 0.03571428571428571] ini
[O15, APelo, -, 1.0, 0.018518518518518517, O37, main, 1.5, 0.012195121951219513] ini
[O44, pridis, -, 1.0, 0.07407407407407407, O44, disprio, 0, 0] ini
[O9, ACheck, -, 1.0, 0.018518518518518517, O35, menu, 0, 0] ini
[O9, ACheck, -, 1.0, 0.018518518518518517, O17, APtrans, 0, 0] ini
[O10, ALtrans, -, 1.0, 0.009259259259259259, O8, AGlo, 0, 0] ini
[O28, Elo, -, 1.0, 0.018518518518518517, O13, main, 0, 0] ini
[O11, main, -, 1.0, 0.018518518518518517, O11, Amenu, 0, 0] ini
[O41, Ptrans, -, 1.0, 0.018518518518518517, O37, main, 0, 0] ini
[O16, APelo, -, 1.0, 0.08333333333333333, O45, main, 1.5, 0.012195121951219513] ini
[O17, APtrans, -, 1.0, 0.06481481481481481, O21, main, 0, 0] ini
[O7, AElo, -, 1.0, 0.009259259259259259, O31, main, 0, 0] ini
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel

[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] vel

Step2: Evaluate the Fitness

Select the best chromosome based on fitness

1.0185185185185186	0.5833333333333334	1.0185185185185186
5.0779244591125785	2.6005291005291005	1.0185185185185186
1.0462962962962963	2.530713640469738	1.0185185185185186
2.0725725725725725	1.074074074074074	2.5955284552845526
1.0185185185185186	1.0185185185185186	1.0648148148148149
1.0185185185185186	1.0185185185185186	1.0092592592592593
2.302226383687058	1.0092592592592593	0.5833333333333334

Stage3: Update the Velocity and Position

Select the P_{best} as g_{best} and update the velocity and position values

20 Velocity size	-0.145	0.0202962962962963
-0.049	-0.008629213483146067	0.0
0.021907407407407403	0.0	0.0
0.0	0.0	-0.195
0.0	0.0	0.023074074074074073
-0.11100000000000002	0.0	0.0
0.016592592592592593	-0.126	0.0
-0.8000000000000002	0.0021851851851851854	-0.094
-0.012356435643564357	-0.576	0.015425925925925926
-0.14	-0.010714285714285714	0.0
0.003185185185185185	-0.041000000000000001	0.0
0.0	0.013999999999999999	-0.133
0.0	-0.27	0.017888888888888888
-0.112	-0.002024390243902439	0.0
0.01374074074074074	-0.155	0.0
-0.218	0.002	-0.10500000000000001
-0.011675675675675677	0.0	0.0
-0.14300000000000002	0.0	-0.294
0.009592592592592594	-0.08000000000000002	-0.0013658536585365857
0.0	0.01594444444444444	-0.046000000000000006
0.0	0.0	0.002074074074074074
-0.132	0.0	0.0
0.015555555555555555	-0.12400000000000003	0.0
0.0	0.01633333333333333	-0.07800000000000001
0.0	0.0	0.009777777777777778
-0.07500000000000001	0.0	0.0
0.015685185185185184	-0.046000000000000006	0.0

[1.0, 1.0, 1.0, -0.049, 0.0, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -0.11100000000000002, 0.0, 1.0, 1.0, -0.8000000000000002, -0.012356435643564357] nv
[1.0, 1.0, 1.0, -0.14, -0.0031666666666666666, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -0.112, 0.0, 1.0, 1.0, -0.218, -0.011675675675675677] nv
[1.0, 1.0, 1.0, -0.14300000000000002, 0.0, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -0.132, 0.0, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -0.07500000000000001, 0.0, 1.0, 1.0, -0.145, -0.008629213483146067] nv
[1.0, 1.0, 1.0, 0.0, -0.011018518518518518, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -0.126, -0.00990740740740741, 1.0, 1.0, -0.576, -0.010714285714285714] nv
[1.0, 1.0, 1.0, -0.04100000000000001, 0.0, 1.0, 1.0, -0.27, -0.002024390243902439] nv
[1.0, 1.0, 1.0, -0.155, -0.005666666666666667, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -0.08000000000000002, 0.0, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -0.12400000000000003, 0.0, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -0.046000000000000006, 0.0017592592592592592, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -0.195, 0.0, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -0.094, 0.0, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -0.133, 0.0, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -0.10500000000000001, -0.012574074074074074, 1.0, 1.0, -0.294, -0.0013658536585365857] nv
[1.0, 1.0, 1.0, -0.046000000000000006, -0.0125, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -0.07800000000000001, 0.0011851851851851854, 1.0, 1.0, 0.0, 0.0] nv
[O14, main, -, 0.951, 0.018518518518518517, O2, count, 0.0, 0.0] nin
[O13, AMtrans, -, 0.889, 0.018518518518518517, O35, menu, 3.1999999999999997, 0.04704950495049505] nin
[O10, ALtrans, -, 0.86, 0.04312962962962963, O46, main, 0.0, 0.0] nin
[O37, Mtrans, -, 0.888, 0.018518518518518517, O35, menu, 0.782, 0.04237837837837838] nin
[O21, witdis, -, 0.857, 0.018518518518518517, O35, menu, 0.0, 0.0] nin
[O19, APlo, -, 0.868, 0.018518518518518517, O10, main, 0.0, 0.0] nin
[O42, Pplo, -, 0.925, 0.018518518518518517, O17, main, 1.105, 0.025078651685393256] nin
[O45, main, -, 0.5, 0.0723148148148148, O45, findprio, 0.0, 0.0] nin
[O12, Delet, -, 0.874, 0.054907407407407405, O1, ABank1, 0.924, 0.024999999999999998] nin
[O15, APelo, -, 0.959, 0.018518518518518517, O37, main, 1.23, 0.010170731707317074] nin
[O44, pridis, -, 0.845, 0.0684074074074074, O44, disprio, 0.0, 0.0] nin
[O9, ACheck, -, 0.9199999999999999, 0.018518518518518517, O35, menu, 0.0, 0.0] nin
[O9, ACheck, -, 0.876, 0.018518518518518517, O17, APtrans, 0.0, 0.0] nin
[O10, ALtrans, -, 0.954, 0.011018518518518518, O8, AGlo, 0.0, 0.0] nin
[O28, Elo, -, 0.8049999999999999, 0.018518518518518517, O13, main, 0.0, 0.0] nin
[O11, main, -, 0.906, 0.018518518518518517, O11, Amenu, 0.0, 0.0] nin
[O41, Ptrans, -, 0.867, 0.018518518518518517, O37, main, 0.0, 0.0] nin
[O16, APelo, -, 0.895, 0.07075925925925926, O45, main, 1.206, 0.010829268292682926] nin
[O17, APtrans, -, 0.954, 0.052314814814814814, O21, main, 0.0, 0.0] nin
[O7, AElo, -, 0.9219999999999999, 0.010444444444444444, O31, main, 0.0, 0.0] nin

0.9695185185185184	0.5723148148148148	0.8235185185185184
4.154568023469014	1.8779074074074074	0.9245185185185185
0.9031296296296296	2.217689250225835	0.8855185185185185
1.730896896896897	0.9134074074074073	2.182588527551942
0.8755185185185185	0.9385185185185184	1.0063148148148149
0.8865185185185185	0.8945185185185185	0.9324444444444444
2.0735971702039118	0.9650185185185185	

[O13, AMtrans, -, 0.889, 0.018518518518518517, O35, menu, 3.1999999999999997, 0.04704950495049505] PB
This process can be continued up to maximum iterations

Step 4: Stop if maximum iterations reached

[1.0, 1.0, 1.0, -38.46970941708035, -0.0020258061257385305, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -10.826620780132703, -0.004014488135639966, 1.0, 1.0, -133.28346769285147, -2.798266758108666] nv
[1.0, 1.0, 1.0, -5.7215921625676485, -7.286723024977166E-4, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -8.900818501514365, -9.911283075949916E-4, 1.0, 1.0, -34.46890205483821, -2.3303930855214783] nv
[1.0, 1.0, 1.0, -11.926487299425936, -0.003360098040782415, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -9.642519520711728, -0.0013126997645417368, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -18.44801559398597, -8.432464348483342E-4, 1.0, 1.0, -28.579586098929212, -1.1815838770437035] nv
[1.0, 1.0, 1.0, -4.991078006003284, -1.9050786459257014, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -21.359839354045754, -0.9406518791668655, 1.0, 1.0, -105.14997126588231, -1.467908112611272] nv
[1.0, 1.0, 1.0, -13.713943551896467, -0.0054949706853327, 1.0, 1.0, -53.29736434911815, -0.7519682434659887] nv
[1.0, 1.0, 1.0, -9.821113257607239, -1.7162909429755113, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -30.54664281886151, -0.0045526568772264325, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -1.6600358494822434, -9.830117929937186E-4, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -9.089850865589856, -0.004272499623245838, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -18.8943057763019, -4.5863434894852294E-4, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -19.503159783835482, -9.101665964294671E-4, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -16.144711723719592, -0.0013877945314070292, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -10.186429853578263, -3.3459292373048264, 1.0, 1.0, -47.71764905079706, -0.5531836295577511] nv
[1.0, 1.0, 1.0, -3.1582259011396125, -1.319917665972488, 1.0, 1.0, 0.0, 0.0] nv
[1.0, 1.0, 1.0, -10.86074273703561, -0.005372462018684108, 1.0, 1.0, 0.0, 0.0] nv
[O14, main, -, 21.25963516998779, 0.0021509189029655824, O2, count, 0.0, 0.0] nin
[O13, AMtrans, -, 5.518432196307798, 2.986122377568821E-4, O35, menu, 68.39896074907045, 1.4384400412216856] nin
[O10, ALtrans, -, 3.078900126559721, 0.015482174700317524, O46, main, 0.0, 0.0] nin
[O37, Mtrans, -, 4.571385776094006, 5.080372638523508E-4, O35, menu, 15.97784036985717, 1.219138462637854] nin
[O21, witdis, -, 6.315700892892442, 5.641573309992363E-4, O35, menu, 0.0, 0.0] nin
[O19, APlo, -, 5.151123017403034, 1.0554453441883466E-4, O10, main, 0.0, 0.0] nin
[O42, Pplo, -, 9.634674470112905, 2.330052420727981E-4, O17, main, 14.660087225026018, 0.563865493879369] nin
[O45, main, -, 2.3461483820232996, 0.9672230104456945, O45, findprio, 0.0, 0.0] nin
[O12, Delet, -, 10.69997717526925, 0.5137223112969806, O1, ABank1, 60.153795393853656, 0.8193560102090038] nin
[O15, APelo, -, 7.490605283477058, 0.004863289647072211, O37, main, 23.459013555692522, 0.36134168914810244] nin
[O44, pridis, -, 5.228391571617731, 0.9134365455336555, O44, disprio, 0.0, 0.0] nin
[O9, ACheck, -, 16.255797228452536, 0.0033553472875998006, O35, menu, 0.0, 0.0] nin
[O9, ACheck, -, 0.8951438824249194, 0.002822553542752069 6, O17, APtrans, 0.0, 0.0] nin
[O10, ALtrans, -, 5.190181299633801, 8.618007350981796E-4, O8, AGlo, 0.0, 0.0] nin
[O28, Elo, -, 10.48541855378265, 2.9710232880642614E-4, O13, main, 0.0, 0.0] nin
[O11, main, -, 9.555797062204135, 0.005495413543037521, O11, Amenu, 0.0, 0.0] nin
[O41, Ptrans, -, 8.74778118236108, 9.185595590317072E-4, O37, main, 0.0, 0.0] nin
[O16, APelo, -, 5.432759951475537, 1.683716622148466, O45, main, 23.78380296118636, 0.31354183397224084] nin
[O17, APtrans, -, 1.6129877744647412, 0.660917148544206, O21, main, 0.0, 0.0] nin
[O7, AElo, -, 5.854263080031273, 0.00932872309894292, O31, main, 0.0, 0.0] nin
****

^^

17.214250972121267	0.7686975323930698	6
final	final	20
71.55623812039079 final	3.9030802648442022	11
2.6589028830107426	final	5
final	8.409048754539391 final	17
23.93324819885647 final	9.953768301770813 final	15
5.613582347243277 final	7.399236895448951 final	16
4.492603658538817 final	30.58937040245529 final	12
23.35116862213337 final	2.204238644103153 final	1
3.5827852594599916	5.0211808421219635	7
final	final	4
56.73151972107732 final	13	18
36.45294729720118 final	19	10
5.395576083431363 final	3	9
14.292042899998599	8	2
final	14	

MODIFIED TESTCASES ARE: 496
BASED ON THE UNIQUE TESTCASES OUR APPLICATION CODE IS REWRITED IN MODIFIED CODE PACKAGE
5365 milliseconds
3535728 bits
BUILD SUCCESSFUL (total time: 8 seconds)

Stopping Criteria Conditions:

1. Terminate when a maximum number of iterations has been exceeded
2. Terminate when acceptable solution has been found
3. Terminate when no improvement
4. When normalized swarm reducing close to zero
5. When objective function close to zero

Merits of PSO:

1. It is based on intelligence
2. Applied to Scientific and Engineering problems
3. No overlapping and mutation
4. PSO adapts real number code

Demerits of PSO:

1. Suffer from partial optimism(in exact about speed and direction)
2. Problem of scattering
3. For energy field it never suits

5. Results and Discussion

In our experiment, we have utilized the GA, PSO for Reducing Interactive Faults Based on Optimal Test Cases in Directed Random Testing**.** By seeing the Table 5.1 the fitness value for the suggested technique verified to be superior to the technique where GA is applied.

Table 5.1. Fitness Comparison for Different Iterations using GA and PSO

Iterations	Fitness values	
	Genetic Algorithm	PSO
25	547	621
50	520	610
75	495	558
100	475	496

The Figure 5.1 given below shows the graphical representation of fitness value using the GA and PSO. In this, the fitness values that are obtained from Genetic algorithm and PSO are being plotted. From the graph it is evident that our proposed method using PSO delivers better fitness value when compared to that of the GA.

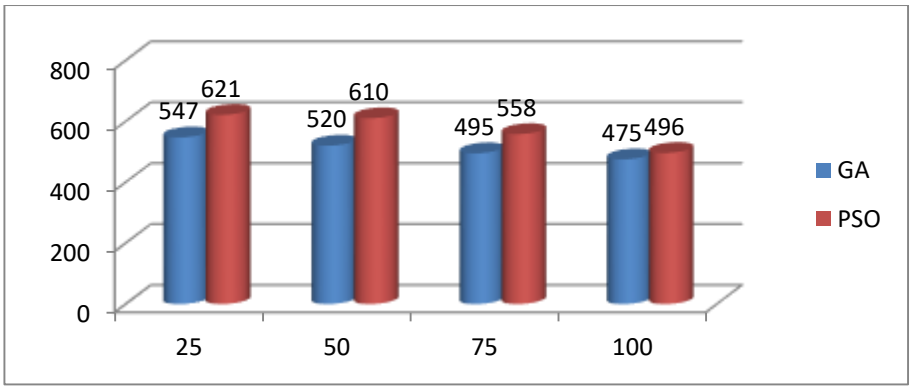


Figure 5.1. Graphical Representation of Fitness Value for Different Iterations of GA, PSO

The Table 2 given below shows the test count value obtained before and after optimization. The test case counts are to be reduced in order to select optimal test cases. The optimized result shows that test cases that are not relevant to the required process are being ignored which is evident from the Test case count obtained after optimization

Table 2. Test Count Value Obtained Before and After Optimization

Iterations	Test case count	
	Without PSO	With PSO
25	661	497
50	661	491
75	661	462
100	661	449

The Figure 5 given below shows the graphical representation of Test count value which is obtained before and after optimization. From the graph it is evident that the Test case count has been reduced to a greater extend when compared to those obtained before optimization.

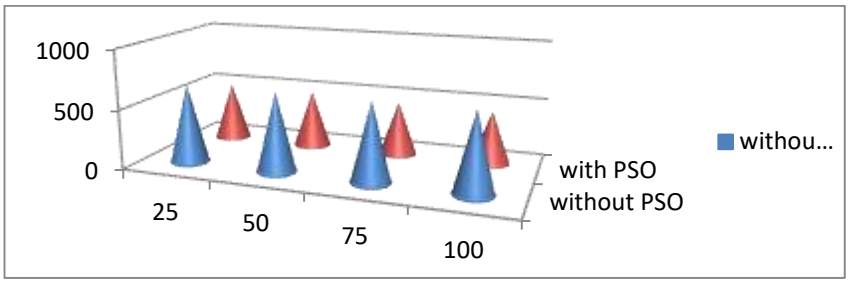


Figure 5.2. Graphical Representation for Test Count Value Obtained Before and After Optimization

The Table 3 given below shows the time and memory usage of our proposed methodology. For each iteration the corresponding time and memory usage are calculated and the results are tabulated. By reducing the interactive faults here we reduce the execution time and memory usage. When the iteration increases, the time usage and memory usage are reduces automatically.

Table 3. Time and Memory Usage for Different Iterations

Iteration	Time usage (milliseconds)	Memory usage (Bits)
25	5214	3699336
50	5365	3535728
75	5368	3289761
100	5741	3146992

The Figure 5.3 and Figure 5.4 given below shows the graphical representation of time and memory usage for different iteration of our proposed technique.

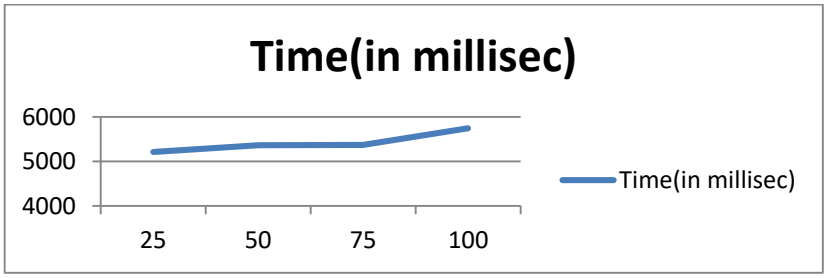


Figure 5.3. Graphical Representation of Time Usage for Different Iterations

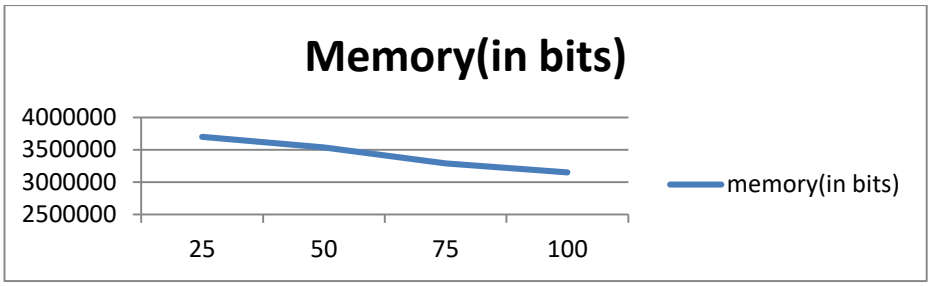


Figure 5.4. Graphical Representation of Memory Usage for Different Iterations

6. Conclusion

In this paper, we have proposed a method for reducing interactive faults based on optimal test cases in directed random testing. The implemented method used to reduce the interactive faults based on Particle Swarm Optimization algorithm. Here, PSO generates the optimal result which will reduce the prohibited inputs. In order to reduce the fault, the proposed method uses the coverage metrics. The result shows that our proposed method remove the ambiguity of randomly generated test cases and produce the optimal result.

References

[1] Z. Quan Zhou, A. Sinaga and W. Susilo "On the Fault-Detection Capabilities of Adaptive Random Test Case Prioritization: Case Studies with Large Test Suites", Hawaii International Conference on System Sciences, **(2012)**, pp. 5584-5593.
[2] J. Kempka, P. McMinn and D. Sudholt, "A theoretical runtime and empirical analysis of different alternating variable searches for search-based testing", Proceeding of conference on Genetic and evolutionary computation conference, ACM, **(2013)**.
[3] J. Pablo Galeotti, G. Fraser and A. Arcuri, "Improving Search-based Test Suite Generation with Dynamic Symbolic Execution", In proceeding of IEEE International Symposium on Software Reliability Engineering (ISSRE), **(2013)**, pp. 360-369.
[4] P. McMinn, "An identification of program factors that impact crossover performance in evolutionary test input generation for the branch coverage of C programs", Information and Software Technology, **(2013)**, pp. 153-172.
[5] G. Fraser, A. Arcuri and P. McMinn, "A Memetic Algorithm for Whole Test Suite Generation", Journal of Systems and Software, **(2014)**, pp. 1-36.
[6] J. Dionisio, T. Mota, I. Pinto and M. Niehus, "Real Time Random Number Generator Testing", Conference on Electronics, Telecommunications and Computers, vol. 17, **(2014)**, pp. 534-541.
[7] X. Niu, Y. Wang and D. Wu, "A Method to Generate Random Number for Cryptographic Application", In proceeding of International Conference on Intelligent Information Hiding and Multimedia Signal Processing, **(2014)**, pp. 235-238.
[8] A. Darvish and C. K. Chang "Black-box Test Data Generation for GUI Testing", In proceeding of IEEE International Conference on Quality Software, **(2014)**, pp. 133-138.
[9] P. Malpani and P. Bassi, "Analytical & Empirical Analysis of External Sorting Algorithms", International Conference on Data Mining and Intelligent Computing, **(2014)**, pp. 1-6.
[10] I. PutuEdySuardiyana Putra and P. Mursanto, "Centroids Based Adaptive Random Testing for Object Oriented Program", Proceeding of IEEE International Conference on Advanced Computer Science and Information Systems, **(2013)**, pp. 39-45.
[11] T. Arts, A. Gerdes and M. Kronqvist, "Requirements on automatically generated random test cases", In Proceedings of IEEE Federated Conference on Computer Science and Information Systems, **(2013)**, pp. 1347-1354.
[12] S. Ali Khan and A. Nadeem, "Automated Test Data Generation for Coupling Based Integration Testing of Object Oriented Programs using Evolutionary Approaches", International Conference on Information Technology, **(2013)**, pp. 369-374.
[13] C. Chow, T. Yueh Chen and T. H. Tse, "The ART of Divide and Conquer An Innovative Approach to Improving the Efficiency of Adaptive Random Testing", In proceeding of IEEE International Conference on Quality Software, **(2013)**, pp. 268-275.
[14] H. Tahbildar and B. Kalita, "Automated Software Test Data Generation: Direction of Research", International Journal of Computer Science & Engineering Survey (IJCSES), vol. 2, no. 1, **(2011)** February, pp. 99-120.

[15] J. Campos, R. Abreu, G. Fraser and M. D. Amorim, "Entropy-Based Test Generation for Improved Fault Localization", IEEE International Conference on Automated Software Engineering (ASE), **(2013)**, pp. 257-267.
[16] B. Yu and Z. Pang, "Generating Test Data Based on Improved Uniform Design Strategy", International Conference on Solid State Devices and Materials Science, vol. 25, **(2012)**, pp. 1245-1252.
[17] L. Padgham, Z. Zhang, J. Thangarajah and T. Miller, "Model-Based Test Oracle Generation for Automated Unit Testing of Agent Systems", IEEE Transactions On Software Engineering, vol. 39, no. 9, **(2013)**, pp. 1230-1244.
[18] B. S. Ahmed, M. A. Sahib and M. Y. Potrus, "Generating combinatorial test cases using Simplified Swarm Optimization (SSO) algorithm for automated GUI functional testing", International Journal an Engineering Science and Technology, vol. 17, **(2014)**, pp. 218-226.
[19] A. Arcuri and L. Briand, "Formal Analysis of the Probability of Interaction Fault Detection Using Random Testing", IEEE Transactions On Software Engineering, vol. 38, no. 5, **(2012)**, pp. 1088-1099.
[20] L. L. Minku, D. Sudholt and X. Ya, "Improved Evolutionary Algorithm Design for the Project Scheduling Problem Based on Runtime Analysis", IEEE Transactions On Software Engineering, vol. 40, no. 1, **(2014)**, pp. 83-102.
[21] J. Lv, H. Hu, K.-Y. Cai and T. Y. Chen, "Adaptive and Random Partition Software testing", IEEE Transactions On Systems Man and Cybernetics: Systems, vol. 44, no. 12, **(2014)**, pp. 1649-1664.
[22] P. McMinn, M. Harman, K. Lakhotia, Y. Hassoun and J. Wegener, "Input Domain Reduction through Irrelevant Variable Removal and Its Effect on Local, Global, and Hybrid Search-Based Structural Test Data Generation", IEEE Transactions On Software Engineering, vol. 38, no. 2, pp. 453-477, **(2012)**.
[23] A. Arcur, "A Theoretical and Empirical Analysis of the Role of Test Sequence Length in Software Testing for Structural Coverage", IEEE Transactions On Software Engineering, vol. 38, no. 3, **(2012)**, pp. 497-519.
[24] K. Koteswara Rao, G. S. V. P. Raju and S. Nagaraj, "Optimizing the software testing efficiency by using a genetic algorithm: a design methodology", ACM SIGSOFT Software Engineering Notes, vol. 38, no. 3, **(2013)**, pp. 1-5.
[25] K. K. Rao and G. S. V. P. Raju, "Developing optimal directed random testing technique to reduce interactive faults-systematic literature and design methodology", Indian Journal of Science and Technology, vol. 8, no. 8, **(2015)**, pp. 715-719.
[26] K. Koteswara Rao and G. S. V. P. Raju, "Theoretical Investigations to Random Testing Variants and its Implications", International Journal of Software Engineering and Its Applications, vol. 9, no. 5, **(2015)**, pp. 165-172.
[27] J. Ratna Kumar, K. Koteswara Rao and D. Ganesh, "Empirical Investigations to Find Illegal and its Equivalent Test Cases using RANDOM-DELPHI", International Journal of Software Engineering and Its Applications, vol. 9, no. 11, **(2015)**, pp. 107-116.
[28] K. Koteswara Rao and G. S. V. P. Raju, "Random Testing: The Best Coverage Technique-An Empirical Proof", International Journal of Software Engineering and Its Applications, vol. 9, no. 12, **(2015)**, pp. 115-122.
[29] K. Koteswara Rao, "Measuring the Function Points from the Points of Relationships of UML", Computer and Electrical Engineering, ICCEE 2008. International Conference on IEEE, **(2008)**.

