Abstract—The objective of software engineering activity is to develop reliable software at a cost effective manner without compromising on its quality. But, software testing comprises of labour-intensive, confusing and error prone actions. To give cost-effective solutions for software test case optimization, issues like classification, minimization, selection, and prioritization are taken into consideration. Many researchers have analyzed about the fitness and optimization of test cases and have arrived at divergent results. This paper using Fuzzy Based Glowworm Swarm Algorithm, discusses about the glowworm, whose objective is to find the areas in the application with highest coverage and usage. This enables to detect failures at an earlier stage of Software Development. The proposed FGSO is a novel algorithm integrating fuzzy logic with the Glowworm Algorithm. This integration eliminates the drawbacks of all standard glowworm algorithms to a considerable degree. The proposed algorithm maximizes the convergence rate of a standard glowworm algorithm by enhancing the performance of its local search. The node with the highest usage that is to be covered by a given test case could be predetermined using this algorithm. Based on the proposed hybrid approach, an optimal result for test case execution is obtained. The performance of the proposed method is evaluated and is compared with other optimization techniques such as ACO, PSO, ABC and PSABC.

Index Terms—Search Domain, Test cases, statement coverage and fault coverage, Optimization, Fuzzy Based Glowworm Swarm Algorithm

I. INTRODUCTION

Software testing is the procedure of identifying errors at the initial stages of development, thereby reducing the cost of fixing bugs before delivering. Though software testing is a very human-intensive, time consuming and expensive process, up until now launching an application without proper testing may lead to cost potentially much higher than that of testing, especially in systems where human safety is involved [1,2]. In general, test cases are the inputs to the program under test. A test case is a group of conditions or variables based on which a tester will find out whether an application or software system is functioning as intended. A pool of test cases in a suite may consist of redundant, extraneous and unfit test cases. Testing becomes an uneconomical process mainly due to the superfluous execution of redundant, irrelevant and unhealthy test cases that increments the burden of cost. The solution is to select the fittest test cases and eliminate the unfit, redundant and needless ones, which in turn leads to test cases optimization [3,4]. Evaluating fitness of test cases is constantly an overwhelming procedure. The term “fitness” defines the correctness of test cases to make sure the quality of software. A test suite is a set of test cases that are designed to test the relevant

DOI: 01.IJRTET.11.1.1570
© Association of Computer Electronics and Electrical Engineers, 2014
functionalities of a software program, ensuring the regular functioning of the application. Multi-criteria test cases fitness evaluation and multi-objective test cases optimization may be two of the serious problems for next generation software testing organizations. The usefulness of this verification and validation procedure depends upon the number of errors originated and corrected before releasing the system. This, sequentially, depends upon the fitness and number of test cases implemented. So, test cases optimization is essential. Optimization techniques have been effectively used in test case generation and prioritization in recent years. Although, a number of optimization techniques has been proposed and good results obtained, problems such as complexity in dynamic data sets, and higher time consumption for convergence exists in the traditional optimization techniques. Thus, there is a scope of improvement to enhance the performance of optimization results. This research work focuses on using appropriate optimization techniques, for test case prioritization which provides optimal results.

The approach here explains the use of swarm intelligence based techniques for test case optimization by prioritizing them. A number of swarm intelligence approaches have been observed to produce significant results in terms of accuracy, convergence behaviour, and time taken. This research uses a couple of recent swarm intelligence approaches for Test Case Optimization [5]. By using fuzzy logic, a novel algorithm called Fuzzy Glowworm Swarm optimization (FGSO) algorithm is introduced here for test case optimization [5]. The proposed algorithm increases the convergence rate of atypical glowworm algorithm by enhancing the algorithm's local search.

II. RELATED WORK

A number of research works have been proposed in the literature for test case selection and prioritization based on optimization process. Some of the well known techniques are discussed here. Regression testing has been solved using optimization approaches like Genetic Algorithm (GA) [6], Ant Colony Optimization (ACO) [7], etc.

Yu-Chi Huang et al. has presented a cost judicious test case prioritization approach with the application of previous data and GA [8]. But the main drawback of this approach is that it does not consider the test cases similarity. Ciyong Chen et al. presented a novel technique called EPDG-GA which uses the Edge Partitions Dominator Graph (EPDG) and Genetic Algorithm (GA) for branch coverage testing [9].

The main drawback of the genetic algorithm is that its convergence rate is very slow compared to other optimization techniques. Moreover, it is more complex since it lacks rank based fitness function which reduces complexity.

ACO is a bio inspired approach based on the real life behavior of ants. Shweta Singhal [10] presented a technique on application of ACO Algorithm for Test Case Selection and Prioritization. This work clearly explains the graphical depiction of food search of the ants which results in finding different paths and choosing the optimal path. Results indicate that ACO results in solutions that are in close proximity with optimal solutions.

ACO approach performs better than GA as convergence is guaranteed, but time to convergence uncertain. Furthermore, in NP-hard problems, high-quality solutions are needed at a faster rate, but ACO concentrates only on quality of solutions.

ArvinderKaur et al. [11] presented the application of Hybrid Particle Swarm Optimization (HPSO) algorithm in regression technique. The criterion considered by ArvinderKaur et al., is maximum fault coverage in minimum execution time. HPSO is an integration of Particle Swarm Optimization (PSO) technique and GA to enhance the search space for the solution. GA offers optimized approach to carry out prioritization in regression testing and on integrating it with PSO technique provides fast solution. GA has been used is Mutation operator which facilitates the search engine to assess all features of the search space. Here, Average Percentage of Faults Detected (APFD) metric has been used to illustrate the significance of HPSO with better transparency.

A large number of fundamental variations have been formulated to enhance the speed of convergence and significance of solution found by PSO. But, fundamental PSO is more suitable to process static, simple optimization problem. Furthermore, it is very difficult to deal with non-metric problem domains in PSO.

ArvinderKaur et al. [12] presented the BCO algorithm for the fault coverage regression test suite prioritization. The most difficult task in regression testing is its time and budget constraints. In bee colony, Scout bees and forager bee are accountable for the development and maintenance of the colony. Based on the nature of these two bees, BCO algorithm for the fault coverage regression test suite has been developed.
BCO algorithm has been designed for fault coverage to obtain maximum fault coverage in minimal execution time of each test case.

III. FORMATION OF FUZZY BASED GLOWWORM SWARM ALGORITHM (FGSO)

This section explains the glowworm swarm optimization and fuzzy logic approach used for test case optimization.

A. Glowworm Swarm Optimization (GSO)

Glowworm swarm optimization (GSO) algorithm has been widely used in solving several optimization problems [13]. In this method, glowworms light emission property has been designed which provided them with prey attraction ability. Light production of glowworms is done by a chemical named Luciferin. Glowworm algorithm is of a great ability in solving problems, such as finding some local optima of multimodal functions simultaneously [14] searching spaces of higher dimensions and orienting multiple sources. In GSO, each glowworm distributes in the objective function definition space [15]. These glowworms carry own luciferin values and have the respective field of idea scope called local-decision range. Their brightness concerns with in the position of objective function value. Brighter the glow, the best is the position that is to say has the good target value. As the glow seeks for the neighbor set in the local-decision range, in the set, a brighter glow has a higher attraction to attract this glow toward this traverse, and the flight direction each time different will change along with the choice neighbor. Moreover, the local-decision range size will be influenced by the neighbor quantity, when neighbor density will be low, glow’s policy-making radius will enlarge favors seeks for more neighbors, otherwise, the policy-making radius reduces. Finally, the majority of glowworm return gathers at the multiple optima of the given objective function.

Each glowworm encodes the object function value \( f(x_i(t)) \) at its current location \( x_i(t) \) into a luciferin value \( l_i \) and broadcasts the same within its neighbourhood. The set of neighbours \( N_i(t) \) of glowworm \( i \) consists of those glowworms that have a relatively higher luciferin value and that are located within a dynamic decision domain and updating by (1) at each iteration.

Local-decision range update:

\[
r_d^i(t + 1) = \min \left\{ r_s, \max \left\{ 0, r_d^i(t) + \beta (n_t - |N_i(t)|) \right\} \right\} ; \tag{1}
\]

and is the glowworm \( i \)’s local-decision range at the \( t + 1 \) iteration, \( r_s \) is the sensor range, \( n_t \) is the neighbourhood threshold, \( \beta \) which affects the rate of change of the neighbourhood range. The number of glow in local-decision range:

\[
N_i(t) = \left\{ j : \|x_j(t) - x_i(t)\| < r_d^i(t); l_j(t) < l_i(t) \right\} ; \tag{2}
\]

and, \( x_j(t) \) is the glowworm \( j \)’s position at the \( t \) iteration, \( l_j(t) \) is the glowworm \( j \)’s luciferin at the iteration.; the set of neighbours of glowworm \( i \) consists of those glowworms that have a relatively higher luciferin value and that are located within a dynamic decision domain whose range \( r_d^i(t) \) is bounded above by a circular sensor range \( r_s \). Each glowworm selects a neighbour with a probability \( p_{ij}(t) \) and process toward it. These movements that are based only on local information, enable the glowworms to partition into disjoint subgroups, reveal an instantaneous taxis-behaviour in the direction and eventually co-locate at the multiple optima of the given objective function. Probability distribution used to choose a neighbour:

\[
p_{ij}(t) = \frac{l_j(t) - l_i(t)}{\sum_{k \in N_i(t)} l_k(t) - l_i(t)} \tag{3}
\]

Movement update:

\[
x_i(t + 1) = x_i(t) + s \left( \frac{x_j(t) - x_i(t)}{\|x_j(t) - x_i(t)\|} \right) ; \tag{4}
\]

Luciferin-update:

\[
l_i(t) = (1 - \rho) l_i(t - 1) + \gamma f(x_i(t)) ; \tag{5}
\]

and \( l_i(t) \) is a luciferin value of glowworm \( i \) at the iteration, \( \rho \in (0,1) \) lead to the reflection of the cumulative kindness of the path followed by the glowworms in their current Luciferin values, the parameter \( \gamma \) only scales the function fitness values, \( f(x_i(t)) \) is the value of test function.
B. Fuzzy Logic

Fuzzy logic establishment is based on fuzzy sets theory. This theory is an overview of traditional sets theory in mathematics. In traditional sets theory, a component fit in to the set or doesn't. Fact that, membership of elements is subsequent to a two value pattern. But, fuzzy sets theory widens this model and applies membership degree. Fuzzy set is defined as:

\[
\{(x, \mu_A(x)) \mid x \in X\}
\]  

(6)

where, the membership of set's members is identified by membership function \(\mu(x)\) that \(x\) represents a specific element and \(\mu\) is a fuzzy function which determines the membership degree of \(x\) in equivalent set and it takes a value between zero and one.

Also, it can be said that \((x)\) is a map from values \(x\) to possible numbers between zero and one. \((x)\) might be a set of discrete or continuous values. The properties uttered to find out fuzzy set's members are fuzzy and are not accurate. Later, it is possible to use different membership functions to illustrate a fuzzy set. Almost, those functions are used which have a simple mathematical representation and are adaptable by a partial number of parameters. However, membership functions are separated into point, linear and nonlinear functions. General form of linear one is motivated from polygonal shapes, like trapezoidal membership function and general form of nonlinear case is owing to bell shapes, like Gaussian membership function. Fuzzy logic is applied widely in solving a variety of problems. Many researchers make use of fuzzy logic to develop optimization algorithms efficiency [16].

IV. METHODOLOGY

In this paper, the standard glowworm algorithm was enhanced by utilization of a fuzzy Gaussian membership function and a new algorithm called fuzzy glowworm swarm optimization (FGSO) was proposed. The proposed FGSO algorithm could develop algorithm convergence to a high extent and increase glowworm algorithm speediness by improving local search. The new-fangled glowworm algorithm, even in some functions, acts much better than PSO algorithm.

In standard GSO algorithm, glowworms have to be familiar with their neighbors before moving step. In this algorithm, a glowworm which needs to alter position toward other glowworms is well thought-out as reference glowworm. After that, other glowworms which their Euclidean distance from this reference glowworm is less than reference glowworm's decision field are measured as reference glowworm's neighbors. This approach has some difficulty. If a glowworm is just a little further than the effect of reference glowworm, it will not present in the neighbours set. Also, it is promising that no other glowworm is in a neighborhood of a glowworm and in moving step, the position of the glowworm with no neighbors doesn't vary. It causes glowworms not to converge to a global optimum with an appropriate rate and to be far away from each other, and even in several cases, a few glowworms might not move until the end of algorithm iterations. According to this case, in order to take into account the effect of more glowworms on each glowworm, the degree of neighborhood of glowworms to each other is firmed by a fuzzy membership function. So neighborhood degree is restored with former idea of crisp neighbors set.

In the proposed methodology each test case would symbolize a Luciferin value of the glowworm and the objective of this method is to find a best glowworms' Luciferin that refers to the test cases with maximum coverage.

The best glowworms' Luciferin value corresponds to a potential solution of the optimization problem and the nectar amount match to the fitness of the associated solution.

A. Proposed Methodology

This research work proposes that the optimized test suite produced by the algorithm will comprise of all possible statements and faults in the program. FGSO is functional to produce an Optimal Test suite by generating optimal test data which would have higher statement and path coverage. The test data will be the required input to be given to the SUT, for travelling along the path and vice versa. At first, the program is given to the Test Case optimization tool, which transforms the corresponding program into an equivalent Control Flow Graph (CFG). The independent paths from the start node to the end node are produced from CFG. Each independent path consists of number of normal nodes and predicate nodes. Every independent path would denote a Test Case. FGSO algorithm is given to produce an Optimal Test suite through optimal test data which would cross the independent paths and then into to the test cases. The search bee would be a search agent which looks for the execution state of the SUT and also initiates the test cases with the initial
test data through equivalence partitioning and boundary value analysis. Then the search agent computes the 
Luciferin value of each test node through evaluating the coverage of each node. This is repeated until an 
executable state of SUT is determined.

Then the glowworm gives the Luciferin value of the traversed nodes/neighboring nodes to the chosen agent 
[11]. The chosen bee evaluates the Luciferin value of traversed nodes and the neighbouring nodes. If the 
Luciferin value of the node obtained is greater than the neighbouring node’s Luciferin value, the node’s 
information is stored in the optimal test case repository. The node whose Luciferin value is observed less is 
discarded.

The algorithm for test case optimization using FGSO algorithm approach is seen below. In order to 
implement any algorithm, the algorithm must be converted into the pseudo code before programmatically 
developing into an application. In the proposed approach, membership degree of glowworm j in the 
neighborhood of glowworm i is an output of a Gaussian function with the input of Euclidean distance 
between glowworms i and j. Center of function is its symmetry axis and finds the position of function model. 

Gaussian membership function is defined as:

$$\mu(x) = \exp\left(-\frac{(x-m)^2}{\sigma}\right)$$

where, m is center and \(\sigma\) is its standard deviation. In improved GSO algorithm by fuzzy logic called fuzzy 
glowworm swarm optimization (FGSO), for every glowworm i, membership degree of other glowworms, j, is 
measured in the neighborhood of i and glowworms with membership degrees less than a threshold are 
removed. In next step, probabilities calculated owing to the differences of glowworms’ Luciferin is multiplied 
by attained membership degrees. For this reason, by changing these probabilities as in (8), additionally to the 
amount of difference of glowworms’ Luciferin, their distance is also efficient in the probability of choosing 
them to move toward each other.

$$p_{ij} \rightarrow \mu_{ij}p_{ij}$$

This is more natural than present situation in GSO algorithm for in certainty also further are glowworms, 
experience less difference of Luciferin and nearer glowworms with lesser Luciferin may have more effect 
than glowworms with more Luciferin but in further distance. Fig. 4 shows pseudo code of FGSO algorithm 
for test case optimization.

Fuzzy glowworm swarm optimization (FGSO) algorithm for Test case Optimization

Set number of dimension = m

Set number of glowworm = n

Initialize the test case which is to be performed by the glowworm

Let s be the step size

Let xi (t) be the location of glowworm i at time t

deploy_agent_randomly,

For i=1 to n do \(l_i(0) = l_0\)

Set maximum iteration number=iter_max

Set t=1;

While (t ≤ iter_max) do:

for each glowworm i do: % Luciferin-update phase

\(l_i(t) = (1 - \rho)l_i(t - 1) + \gamma f(x_i(t))\),

for each glowworm i do: % Movement -phase

\(N_i(t) = \{j: \|x_j(t) - x_i(t)\| < r^i(t)\} \}

\(\mu(x) = \exp\left(-\frac{(x-m)^2}{\sigma}\right)\)

For each glowworm in \(N_i(t)\) do:

\(p_{ij}(t) = \frac{l_j(t) - l_i(t)}{\sum_{k \in N_i(t)} l_k(t) - l_i(t)}\)

\(p_{ij} \rightarrow \mu_{ij}p_{ij}\)

j=select_glowworm (\(\bar{p}\))
\[ x_i(t+1) = x_i(t) + s \frac{x_i(t) - x_i(t)}{\|x_i(t) - x_i(t)\|} \]
\[ r_d^2(t)(t+1) = \min \{ r_x, \max(0, r_d^2(t) + \beta nt - Nt) \} \]
\[ t \leftarrow t + 1; \]

This increases local search property and using fuzzy neighborhood increases convergence rate algorithm speed generally. FGSO iterates till its stopping criterion is met which determines the maximum number of paths covered and faults covered.

To implement the above algorithm, the proposed approach uses the Test Suite Optimization tool to optimize the Test Cases by employing the FGSO algorithm. The tool considers a program as an input to generate independent paths. Using the generated independent paths Test Cases are traversed along the paths with the help of FGSO algorithm. By doing so, the test cases with maximum coverage (High optimal Value) are recognized. Finally the optimal Test Suite is generated as an output.

V. RESULTS AND DISCUSSION

This section briefly explains about the performance of the proposed FGSO approach. Here, initially the performance of the proposed optimization algorithm is measured using general optimization parameters. Then the simulation results for test case optimization using proposed algorithm is shown to evaluate the effectiveness of the proposed FGSO approach.

A. Simulation Results

The experiment is implemented in MATLAB. The test case prioritization technique’s basic evaluation is to have maximum number of faults covered and statement covered with minimum number of test cases required. In this approach, the execution time of every test case is also analyzed. The fault measuring technique used is fault coverage based testing technique. In this example, there are test cases forming Test Suite (TS) = \{T1, T2, T3, T4, T5, T6, T7, T8, T9, T10\} and the faults covered by those test cases are represented as Faults Covered (FC) = \{F1, F2, F3, F4, F5\}. Similarly the statements covered by the test cases are denoted as Statements Covered (SC) = \{S1, S2, S3, S4, S5, S6, S7\}. The Control Flow Graph (CFG) is seen in Fig 1.

![Figure 1.CFG of the quadratic equation code](image)

Table 1 and 2 clearly shows the Test cases with the faults and statements covered in particular execution time.

<table>
<thead>
<tr>
<th>Test Case/Faults</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>No. of Faults Covered</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>T2</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>T3</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>T4</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>T5</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>T6</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>T7</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>T8</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>T9</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>T10</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
This section compares the performance of the proposed FGSO approach with the other optimization approaches such as ACO, PSO, ABC and PSABC in terms of percentage of statement coverage and fault coverage.

<table>
<thead>
<tr>
<th>Test Case/Faults</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
<th>No. of Faults Covered</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>3</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>T2</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>T3</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td>3</td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>T4</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>T5</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>5</td>
<td></td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>T6</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>2</td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>T7</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td>3</td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>T8</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td>3</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>T9</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td>3</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>T10</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 2. No. of Cycles Vs Statement Coverage (%) Comparison

Fig 2 shows the comparison of the number of cycles and the statement coverage in percentage. From the fig 2 that the proposed test case prioritization approaches using FGSO provides better statement coverage when compared with ACO, PSO, ABC and PSABC optimization approaches. When the number of cycles increases, the Statement coverage also increases linearly. For instance, when the number of cycles is 14, the statement coverage of the ACO, PSO, ABC and PSABC approach is 31%, 40%, 47% and 51% respectively. But, when the proposed FGSO approach is considered, the statement coverage attained is 56%. Similarly, for the other cycles, the proposed FGSO based Test case prioritization approach provides better results when compared with the other existing approaches.

Figure 3. No. of Cycles Vs Fault Coverage (%) Comparison
Fig 3 shows the fault coverage comparison in percentage for the approaches such as ACO, PSO, ABC and PSABC. It can be observed from the graph that, there is significant increase in the faults coverage with the increase in the number of cycles. The proposed approach outperforms the other two approaches in terms of the fault coverage. For example, considering the number of cycles is 14, the fault coverage obtained by the approaches like ACO, PSO, ABC and PSABC are 42%, 54% and 59% respectively. But, when the proposed FGSO approach is considered, the fault coverage obtained is 65% which is higher than the approaches taken for consideration. Thus, it can be observed from the simulation results that the test cases are prioritized based on higher statement coverage and fault coverage using the FGSO approach.

**APFD Metric**

To quantify the goal of increasing a subset of the test suite's rate of fault detection, we use a metric called APFD developed to measures the rate of fault detection per percentage of test suite execution [17]. The APFD is calculated by taking the weighted average of the number of faults detected during the run of the test suite. APFD can be calculated as follows:

$$ APFD = 1 - \frac{(Tf_1 + Tf_2 + \cdots + Tf_m + \frac{1}{2n})}{nm} $$  \hspace{1cm} (9)

Where n be the no. of test cases and m be the no. of faults. \((Tf_1, \cdots, Tf_m)\)are the position of first test T that exposes the fault.

![Figure 4. Performance Comparison of APFD measure for test cases](image)

Fig 4 shows the comparison of APFD measures for various proposed approaches such as ACO, PSO, ABC and PSABC. It can be observed from the graph that, there is significant increase in the percentage of APFD measures. The proposed FGSO approach outperforms the other approaches in terms of the APFD measures.

V. CONCLUSIONS

The aim of the software industry is to make sure the availability of high quality software to the end user. Testing ensures that the software meets the user conditions and necessities. Effectual generation of test cases and prioritization of test cases has to be addressed in the field of Software Testing. Factors like effort, time and cost of the testing are factors influencing these. A number of research works have been proposed in the literature for test case prioritization. The main aim for prioritization of test cases is to minimize the cost and time of regression testing. The objectives considered in this research work are statement coverage and fault coverage within a minimum execution time. This research work aims in attaining test case prioritization results using Fuzzy Based Glowworm Swarm Algorithm (FGSO). It is observed from the experimental results that the proposed approach based test case prioritization based approach provides better results when compared with ACO, PSO, ABC and PSABC.

REFERENCES


[7] Hyeon-Cheol Jo ; Kwang-SeonYoo ; Jae-Yong Park ; Seog-Young Han, “Dynamic topology optimization based on ant colony optimization”, *Eighth International Conference on Natural Computation (ICNC)*, 2012


