International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 9 (2017) pp. 1920-1928
© Research India Publications. http://www.ripublication.com

# Optimal Test Suite Selection using Improved Cuckoo Search Algorithm Based on Extensive Testing Constraints

**[1]K. Senthil Kumar**
*[1]Research Scholar, Research and Development Centre,*
*Bharathiar University, Coimbatore – 641 046, Tamil Nadu, India.*
*Assistant Professor (Sr. G), Department of Information Technology,*
*Faculty of Engineering and Technology, SRM University, Chennai*

**[2]Dr. A. Muthukumaravel**
*[2]Professor & Head, Department of MCA,*
*Bharath University, Chennai, Tamil Nadu, India.*

Abstract: Software engineering deals with the development of software systems and to reduce the cost and improve the development process. Software metric is a quantitative measure of a degree to which a software system or process some property. It is consider as the major basis as it depends on the software quality. In this paper we propose a method for estimation of software quality based on the software metrics. We make reuse of software in order to reduce effort, time and cost and thus it increases the productivity and quality of software programs. In order to develop better quality software, we have considered as software metrics like reusability, fault proneness and change proneness. In our proposed method test cases are optimized by using Particle Swarm Optimization algorithm (PSO). Then the optimized test cases are prioritized by using Improved Cuckoo Search algorithm (ICSA). Finally, the optimized result will be evaluated by software quality measures.

**Keywords:** Particle Swarm Optimization, Cuckoo Search, Software metrics, Quantitative measure.

## INTRODUCTION

For attaining high quality software, Software testing is one of the major and primary techniques. Software testing is prepared to identify presence of errors, which cause software failure [14]. Test case prioritization is an efficient and practical technique of regression testing. It is useful to increase the competence of regression testing by sorting and implementing test cases according to their significance [5]. Test case prioritization has also been employed to decrease the quality assurance cost and for minimizing the fault detection effort [2]. Majority of the presented test case prioritization techniques are based on particular coverage criteria. That is based on particular performance goal such as fault coverage, statement coverage, path coverage, branch coverage, and function coverage [3] the test cases are prioritized. Regression testing is the process of authenticating modifications brought in a system during software maintenance. It is an expensive, still an imperative process. As the test suite size is very huge, system retesting consumes huge amount of time and computing resources [6]. In regression testing the test case prioritization is significant. It programs the test cases in a regression test suite with a view to maximizing particular objectives which assist to reduce the time and cost necessary to maintain service-oriented business applications. Presented regression testing techniques for such applications focus on testing individual services or workflow programs [9]. Regression test prioritization means arrangement of the test cases in an increasing or decreasing order to meet some presentation goal.

Different prioritization criteria may be employed to the regression test suite with the objective of meeting those criteria [13]. Experimenting and validating the part of code are the activity carried out inside dissimilar phases. Tasks of regression testing are Test Case Prioritization, Test Suite Selection, Test case reduction which present the guarantee that no intended fault is generated while altering the code [12]. For example, software development method based on components mostly employs the black-box components, frequently admit from third party. The internal of third party components are unfamiliar to the users, when any changes are produced in third party components might get in the time way with other software system [11]. While the significance of incorporating requirements information during the testing phase has been well known by the requirements engineering community only some researchers have studied the employ of requirements with software testing (black-box testing) [8]. To build the rate of deficiency location experiment prioritization is the process of requesting the implementation of experiments. Expanding the rate of fault discovery can be more input to framework engineers, improving obligation create action and, eventual, programming conveyance. Several presented experiment prioritization procedures admit that tests can be carried out in any request [10].

Employing code coverage information as substitute's coverage based test case prioritization reorganizes test case in ordered to exploit code coverage as early as feasible. On the other hand code coverage itself is not an ample criterion to guarantee the accomplishment of high rate of faults detection

[4]. Several test case prioritization techniques have been suggested and most are based mainly on structural coverage metrics such as branch or statement coverage [1]. There are several algorithms employed for test case prioritization such as greedy, additional greedy, hill climbing, Additional Greedy Algorithm, Heuristic Algorithm, Genetic Algorithm and particle swarm optimization. Each algorithm is containing their own functionality. The algorithms will vary in terms of the presentation [15].

## LITERATURE SURVEY

Hong Me et al [16] have proposed an approach to prioritizing test cases in the absence of coverage information that operating on Java programs tested under the JUnit framework an increasingly popular class of systems. The JUnit test case Prioritization Techniques operating in the Absence of coverage information (JUPTA), analyzing the static call graphs of JUnit test cases and the program under test to estimate the ability of each test case to achieve code coverage, and then scheduling the order of these test cases based on those estimates. They evaluate the effectiveness of JUPTA, they were conducting an empirical study on 19 versions of four Java programs ranging from 2K-80K lines of code, and compared several variants of JUPTA with three control techniques, and several other existing dynamic coverage-based test case prioritization techniques, assessing the abilities of the techniques to increase the rate of fault detection of test suites.

Bestoun S. Ahmed et al [17] have proposed a strategy for GUI functional testing using Simplified Swarm Optimization (SSO) was proposed. The SSO used to generate an optimized test suite with the help of Event-Interaction Graph (EIG). The proposed strategy also manages and repairs the test suites by deleting the unnecessary event sequences that are not applicable. The proposed generation algorithm based on SSO has proved its effectiveness by evaluating it against other algorithms. In addition, the strategy was applied on a standard case study and proved its applicability in reality.

Ke Zhai et al [18] have proposed a suite of metrics and initialize them to demonstrate input-guided techniques and point-of-interest (POI) aware test case prioritization techniques, differing by whether the location information in the expected outputs of test cases is used. It reports a case study on a stateful LBS-enabled service. The case study shows that the POI-aware techniques can be more effective and more stable than the baseline, which reorders test cases randomly, and the input-guided techniques. They also find that one of the POI-aware techniques, cdist, was either the most effective or the second most effective technique among all the studied techniques in their evaluated aspects, although no technique excels in all studied SOA fault classes.

Sreedevi Sampath et al [19] have proposed a formalized the notion of combining multiple criteria into a hybrid. Their goal was to create a uniform representation of such combinations so that they can be described unambiguously and shared among researchers. They envision that such sharing will allow researchers to implement, study, extend, and evaluate the hybrids using a common set of techniques and tools. They precisely formulate three hybrid combinations, Rank, Merge, and Choice, and demonstrate their usefulness in two ways. First, they were focusing recast, in terms of their formulations; others' previously reported work on hybrid criteria. Second they use previous results on test case prioritization to create and evaluate new hybrid criteria. Their findings suggest that hybrid criteria of others can be described using their Merge and Rank formulations, and that the hybrid criteria they developed most often outperformed their constituent individual criteria.

Ashima Singh [20] has proposed a time overhead-based criteria to -prioritize test cases. The approach was providing the solution of test cases sequencing as well as reduction by using an intelligent dynamic approach. They generate the test cases based on the priorities, which are assigned by the algorithm to test cases on the basis of some intelligent operations. A cumulative mutation probability (CMP) metric was used to determine the effectiveness of the new test case orderings.

Ahlam Shakeel Ahmed Ansari et al [21] have proposed a technique to deliver a quality product with decreased regression testing time and cost and when rigorous regression test was performed with reduced test suites it should produce the same quality product as the original test suites would produce. In order to address the above problem, two phase test suite refinement approach is proposed. The two phases of the proposed approach are Test Suite Minimization and Test Suite Prioritization. In Test Suite Minimization specification based minimization technique was used, whereas in Test Suite Prioritization risk based prioritization technique was used. That approach was enable thus to achieve quality product with decreased regression testing time and cost, as it was generating the reduced test suites which will be equivalent to the original test suited for the product.

Gurinder Singh et al [22] have proposed an new test case reduction hybrid technique based on Genetic algorithms(GA) and Ant colony optimization (ACO). GA was an evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover. ACO was a swarm intelligence algorithm. They adopts the behavior of ants to solve the given problem. It proved to be optimistic approach which provides optimum results in minimum time.

## PROBLEM DEFINITION

- ❖ Examining test results physically can be extremely unwieldy and time consuming for test suits.
- ❖ A breakdown of a publishing error may take place if the explanation of the service is wrong or the consumption is problematic.
- ❖ To decrease the quality assurance cost which comprises both the testing and debugging cost while minimizing the loss of diagnostic fault information.
- ❖ Checking scenarios are made difficult if multiple test cases are employed in mixture with multiple components.

- ❖ Number of errors identified during non-overlapping intervals is independent of each other.
- ❖ Alteration probabilities affect the system dependability estimations drastically.
- ❖ The problem with decreasing fault detection effort was that it may cause the information loss, thus of which clearing up cost gets raise.

## PROPOSED METHODOLOGY

Software metrics are proposed to work out the software quality based on the Improved Cuckoo search Algorithm is the main intension of the suggested method. At first the test cases are produced from the application program. After that, the quality based features are removed from the test cases the features are Fault and Execution Time. Next we employ Particle Swarm Optimization (PSO) algorithm to optimize the removed features. After that another time the features are removed from that optimized features they are Cohesion, Coupling, If value, then these features are optimized by employing Improved cuckoo Search Algotihm and presentation features quantitatively, met during the planning and execution of software development. For our purpose of reusability, fault proneness and change proneness are calculated approximately as testing constraints these can offer as to measures of software product. In Fig.1 the specified process of implemented method is revealed. In Fig, 1the block diagram of the proposed method is revealed.

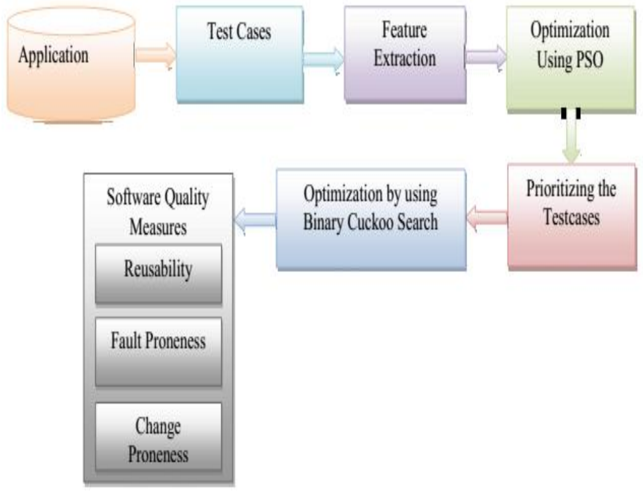

**Figure 1:** Block Diagram of Proposed method

### *A. Test Case Generation*

In the application, Test cases are used to study all the feasible combination. Test case generation is a method where the test cases are produced not based on an algorithm but based on the ones statement of the application. Classes will be made certain and different test inputs will be offered to make certain for the faults in the application. Currently, the test cases are generated based on the application. Features are removed from the test cases. It is made cleared in the beneath section.

### *B. Feature Extraction*

To evaluate and forecast the software quality the features are removed from the test cases. These features based software that includes a strong relationship with software quality. The characteristics are Fault and Execution Time. By using a particle Swarm Optimization Algorithm to optimize these attributes to choose the best test cases.

### *C. Optimization Using Particle Swarm Optimization (PSO)*

Population based search protocol may turn out to be Particle swarm Optimization (PSO). PSO was designed on the basis of the social presentation of birds in a flock. Each particle flies in the search space with a velocity varied by its own flying memory and its companion's flying experience in PSO. Each particle has its major function value which is settled on by a fitness function. PSO is an evolutionary algorithm which is much connected to that of the Genetic Algorithm where an exact system is initialized by a population of random solutions. It actually is made to assist pretend the good manners concerning birds all through hunt for foodstuff by means of a cornfield or even fish institution. The technique can successfully come across best or even around best remedies all through big search rooms. In flowchart Fig.2 the overall process is revealed.

**"Individual best":** It is the individual best choice algorithm by assessing each individual position of the particle to its own best position $pbest$, only. The data about the other particles is not used in this $pbest$.

**"Global best":** It is the universal best selection algorithm, which obtains the global information by making the movement of the particles encloses the position of the best particle from the complete swarm. Additionally, every particle uses its experience with earlier incidents in terms of its own best solution.

- ➢ **Swarm initialization**: For a population size u, arbitrarily generate a solution.
- ➢ **Define the fitness f unction**: According to the current solution, the fitness function chosen should be used for the constraints.
- ➢ $gb$ **and** $pb$ **Initialization:** In the start the fitness value estimated for every particle is placed as the Pbest value of each particle. Among the Pbest values, the optimal one is chooses as the $gb$ value.
- ➢ **Velocity Computation:** The new velocity is calculated using the beneath equation

$$V_i^{d+1} = V_i^d + c_1.r_1.(pb_i^d - x_i^d) + c_2.r_2.(gb^d - x_i^d) \quad (2)$$

$$x_i^d = x_i^d + \delta V_i^d \quad (3)$$

In eq. (2),

$r_1, r_2$ - independent random numbers generated in the range [0-1]

$V_i^d$ - Velocity of ith particle

$x_i^d$ - Current position of the particle $i$

$pb_i^d$ - Best fitness value of the particle at the current iteration

$gb^d$ - Best fitness value in the swarm.

- **Swarm Updation:** Find out the fitness function once more and improve the $pb$ and $gb$ values. If the new value is better than the earlier one, replace the old by the current one. And as well select the optimal $pb$ as the $gb$.
- **Criterion to stop:** Extend till the solution is really appropriate or maximum iteration is attained

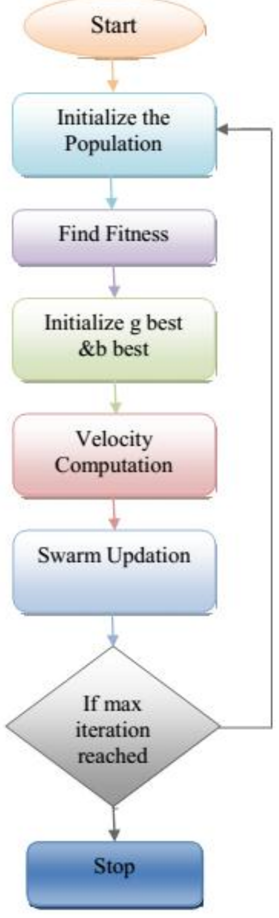

**Figure 2:** Flow chart for Particle Swarm Optimization Algorithm

*C. Prioritizing the Test Cases*

Prioritizing the test cases the features are removed from the PSO optimized test cases the features are Cohesion, Coupling, If value.

*i. Cohesion*

Cohesion is an ordinal category of measurement and is frequently explained as "high cohesion" or "low cohesion". Modules with high cohesion tend to be preferable since high cohesion is related with numerous enviable features of software including robustness, reliability, reusability, and understandability whereas low cohesion is related with undesirable features such as being hard to uphold, test, reuse, and even comprehend.

*ii. Coupling*

Coupling is the way and degree of interdependence among software modules; a measure of how directly linked two routines or modules are the strength of the relationships among modules. Coupling is frequently compared with cohesion. Low coupling frequently associates with high cohesion, and vice versa. Low coupling is frequently a sign of a well-structured computer system and a good design, and when joined with high cohesion, supports the common goals of high readability and maintainability.

*iii. If Value*

Here we are assuming the probability value are, If any function contains "if" condition then allocates 0.5 values in the test case or else assign one. If the features are extracted form the PSO optimized test cases. The features are Cohesion and Coupling.

*D. Modified cuckoo search algorithm*

Cuckoo search algorithm is a metaheuristic algorithm which was inspired by the breeding behavior of the cuckoos and alleviates to implement. There are a number of nests in cuckoo search. Each egg points out a solution and an egg of cuckoo indicates a new solution. The new and better solution is replacing the most awful solution in the nest. The subsequent representation scheme is selected by Cuckoo Search algorithm: Each egg in a nest symbolizes a solution, and a Cuckoo egg symbolizes a novel solution. The plan is to employ the novel and probably better egg to substitute a not-so-good egg of Cuckoo in the nests. On the other hand this is the fundamental case i.e., one cuckoo per nest, but the extent of the approach can be raised by incorporating the property that each nest can have more than one egg which symbolizes a set of solutions. The process of clustering is given beneath,

- At a time only one egg is laid by cuckoo. Cuckoo dumps its egg in a arbitrarily selected nest.
- The number of accessible host nests is fixed, and nests with high quality of eggs will transmit over to the next generations.
- In case of a host bird found out the cuckoo egg, it can throw the egg away or discard the nest, and build a totally novel nest.

**Step 1: Initialization Phase**

The population ($m_i$, where i=1, 2,…n) of host nest is started arbitrarily.

**Step 2: Generating New Cuckoo Phase**

Using levy flights a cuckoo is selected at random and it produces new solutions. After that the produced cuckoo is evaluated using the objective function for finding out the quality of the solutions.

**Step 3: Fitness Evaluation Phase**

Evaluate the fitness function based on the equation and next select the best one.

$$fitness = \max imum\ \ popularity \tag{4}$$

**Step 4: Updation Phase**

The superiority of the new solution is evaluated and a nest is selected among arbitrarily. If the excellence of new solution in the selected nest is better than the old solutions, it will be replaced by the new solution (Cuckoo). Otherwise, the previous solution is placed aside as the best solution. The levy flights employed for ordinary cuckoo search algorithm is,

$$m_i^* = m_i^{(t+1)} = m_i^{(t)} + \alpha \oplus Levy(n) \qquad (5)$$

$$\begin{cases} \text{if } S < rand \text{ then } m_i^{(t+1)} = 0 \\ \text{if } S > rand \text{ then } m_i^{(t+1)} = 1 \end{cases}$$

Where $t$ is step size, and $\alpha > 0$ is the step size scaling factor/parameter. Here the entry wise product $\oplus$ is similar to those used in PSO, $x_i^{(t+1)}$ and stands for $(t+1)_{th}$ egg (feature) at nest (solution), *i=1,2,....m*, and *t=1,2,...d*. The L´evy flights employ a random step length which is drawn from a L´evy distribution. Therefore, the CS algorithm is more efficient in exploring the search space as its step length is much longer in the long run

In traditional COA, the solutions are revised in the search space towards continuous-valued positions. Not like, in the BCOA for feature selection, the search space is modeled as a dimensional boolean lattice, in which the solutions are revised across the corners of a hypercube. Additionally, as the problem is to choose or not a given feature, a solution binary vector is used, where 1 corresponds whether a feature will be chosen to compose the new dataset and 0 otherwise. In order to build this binary vector, we have employed the equation 4, which can offer only binary values in the boolean lattice limiting the novel solutions to only binary values:

$$S\left(m_i^{(t+1)}\right) = \frac{1}{1+e^{-m_i^{(t)}}} \qquad (6)$$

**Step 5: Reject Worst Nest Phase**
The worst nests are thrown away in this part, based on their chance values and new ones are built. Presently, function the best solutions are ranked based on their fitness. After that the best solutions are identified and spotted as optimal solutions.

**Step 6: Stopping Criterion Phase**
Till the maximum iteration achieves this process is repeated. The optimized effect will be inspected for the measure of software quality. The précised process is clearly showed in flowchart. It's revealed in beneath.

*E. Testing Constraints*
Quality of software is more and more significant and testing connected issues are turning into crucial for software. Methodologies and techniques for forecasting the testing effort, watching process costs, and measuring effects can assist in increasing competence of software testing. For prioritizing the test cases, reusability, fault proneness and change proneness are calculated approximately as testing constraints.

*i. Reusability*
Reusability is the degree to which an element can be reused and decreases the software improvement cost by enabling less coding and more interration. The reusability of benefits is dissimilar in dissimilar contexts. On the other hand, there are some features that commonly supply to the reusability of assets. Even though several of these features apply to benefits in common, we spotlight in this section, we spotlight on components as assets. At a high level, we differentiate two features of reusability.

$$\mathrm{Re}usability = Usability + Useful\ln ess$$

Usability is the level to which a benefits is easy to use in the sense of the amount of effort that is required to use an asset. Usability is the frequency of suitability for helpfulness depends on the functionality the generality and quality of an element.

*ii. Fault Proneness*
In software engineering Fault-prone modules prediction is one of the most customary and significant areas. Finding of fault-prone modules has been extensively studied Fault Proneness is employed to calculate the software can be key step towards steering the software testing and enhancing the efficiency of the whole process.

*iii. Change Proneness*
Series of alters are made to software during software evolution. Alterations can be due to a mixture of reasons such as enhancements, adaptation, perfective maintenance or fixing defects. Several parts of the software may be more prone to changes than others. Knowing which classes are changes prone can be very supportive; change-proneness may point out particular underlying quality issues.

## RESULTS AND DISCUSSION

The experimental result of improved cuckoo search algorithm is discussed below. The proposed system is implemented using MATLAB 2014 and the experimentation is performed with i5 processor of 3GB RAM.

*A. Performance Evaluation*
Performance Evaluation is a multi-purpose tool used to: Measure actual performance against expected performance. Provide an opportunity for the employee and the supervisor to discuss job performance. Identify employee training and development needs, and plan for career growth.

**Table I:** Fitness value depending on iteration

| No of Iterations | Fitness value using Cuckoo Search Algorithm |
|---|---|
| 5 | 12.658 |
| 10 | 12.658 |
| 15 | 11.984 |
| 20 | 10.986 |
| 25 | 10.986 |

Below specified fig.3 explains the outcomes of the fitness value based on the iteration.

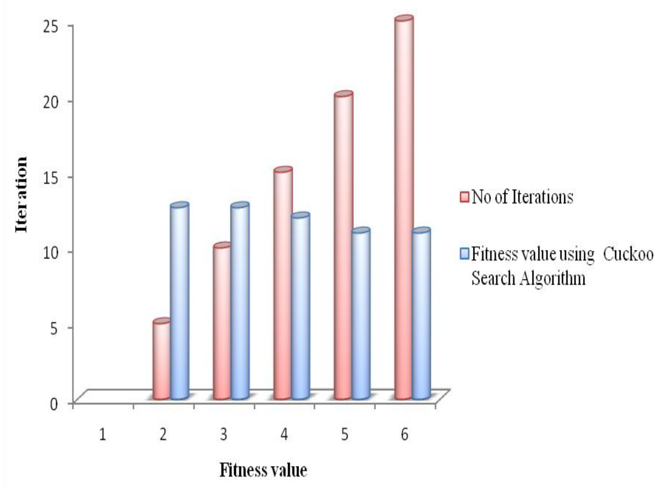

**Figure 3:** Graph for fitness values based on iterations

From table I, The results of the fitness value using cuckoo search measures are graphically represented in fig. 3. The fitness value of the first iteration is 5 the fitness value is 12.658 then iteration 10 the fitness values are 11 in iteration 15 the fitness values are 11.984 , iteraion 20 and 25 are have a same fitness value 10.986.

**Table II:** Time depending on no of iteration

| Time | No of iteration |
|---|---|
| 25698 | 5 |
| 26845 | 10 |
| 27668 | 15 |
| 28964 | 20 |
| 30689 | 25 |

Below specified fig.4 explains the outcomes of the Time based on the iteration.

From table II, The results of the Time value using cuckoo search measures are graphically represented in fig. 4. TheTime value of the first iteration is 5 the Time value 25698then iteration 10 the Time values are 26845 in iteration 15 the Time values are 27668, iteraion 20 then the Time value is 28964, iteration 25 then the Time value is 30689.

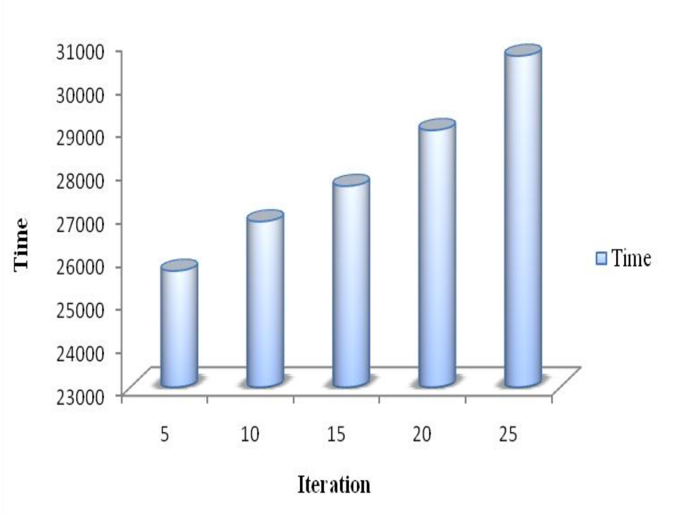

**Figure 4:** Graph for Time based on iteration

**Average Percentage of Faults Detected (APFD) Metric**
To calculate the goal of increasing a subset of the test suite's rate of fault detection, we employ a metric called APFD. That calculates the rate of fault detection per percentage of test suite implementation. The APFD is computed by taking the weighted average of the percentage of faults detected during the implementation of the test suite. APFD values range from 0 to 100; higher values imply faster (better) fault detection rates. APFD can be computed as follows:

$$APFD = 1 - \{(Tf1 + Tf2 + \ldots + Tfm)/mn\} + (1/2n)$$

Where n be the no of test cases and m be the no of faults. ($Tf1, \ldots, Tfm$ ) are the position of first test T that exposes the fault.

**Table III:** Result of the proposed APFD based on iteration

| **No of Iteration** | **APFD** |
|---|---|
| 5 | 0.7589 |
| 10 | 0.7589 |
| 15 | 0.7589 |
| 20 | 0.7589 |
| 25 | 0.7589 |

Below specified fig.5 explains the outcomes of the APFD value based on the iteration.

From table III, The results of the APFD value measures are graphically represented in fig. 5. The APFD value of the first iteration 5 then the APFD value is 0.7589 then iteration 10 the APFD values are 0.7589 in iteration 15 the APFD values are 0.7589, iteraion 20 then the APFD value is 0.7589, iteration 25 then the APFD value is 0.7589.

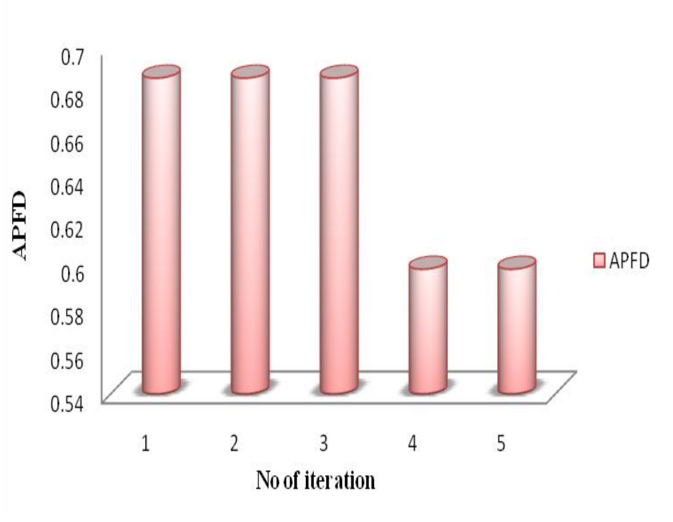

**Figure 5:** Graph for APFD performance

**Problem Tracking Reports (PTR) Metric**

$$PTR(t,p) = nd/n$$

Let t - be the test suite under assessment,
n- the total number of test cases in the total number of test cases required to identify all faults in the program under test p.

**Table IV:** Result of the Proposed PTR based on iteration

| No of Iteration | PTR |
|---|---|
| 5 | 0.6854 |
| 10 | 0.6854 |
| 15 | 0.6854 |
| 20 | 0.5974 |
| 25 | 0.5974 |

From table IV, The results of the PTR value measures are graphically represented in fig. 5. The PTR value of the first iteration 5 then the PTR value is 0.6854then iteration 10 the PTR values are 0.6854in iteration 15 the PTR values are 0.6854, iteraion 20 then the PTR value is 0.5974, iteration 25 then the PTR value is 0.5974.

Below specified fig.6 explains the outcomes of the PTR value based on the iteration.

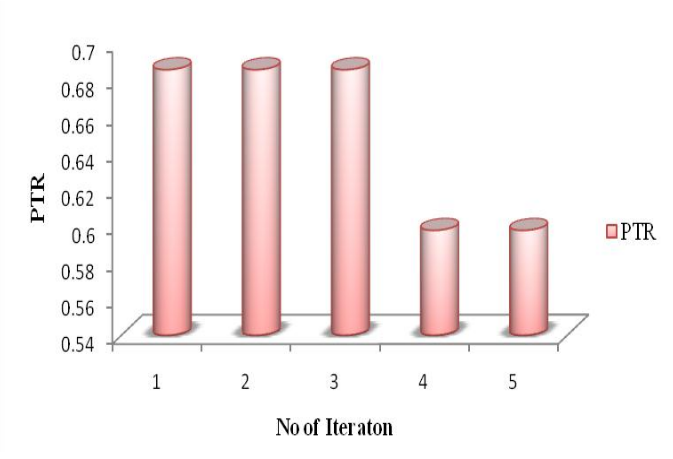

**Figure 6:** Graph for PTR performance

**Table V:** Result of the Memory space depending on the iteration

| Iteration | memory |
|---|---|
| 5 | 2463456 |
| 10 | 2478956 |
| 15 | 2497456 |
| 20 | 2568874 |
| 25 | 2578955 |

Below specified fig.7 explains the outcomes of the Memory based on the iteration.

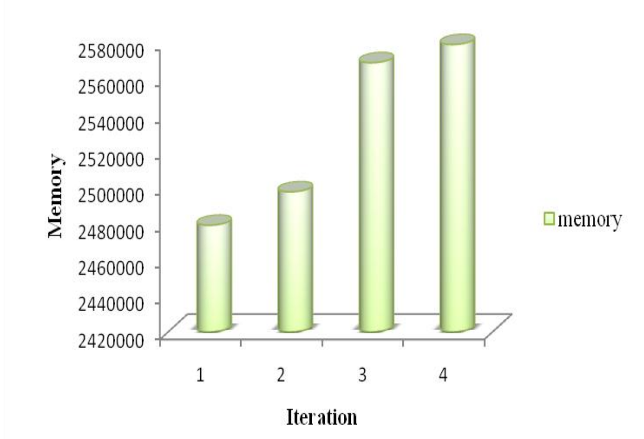

**Figure 7:** Graph for memory based on iteration

**Table VI:** Result of the Before and After Prioritization

| Before Prioritizati on | After Prioritizati on |
|---|---|
| 97% | 91% |

Below specified fig.8 explains the outcomes of the before and After prioritization.

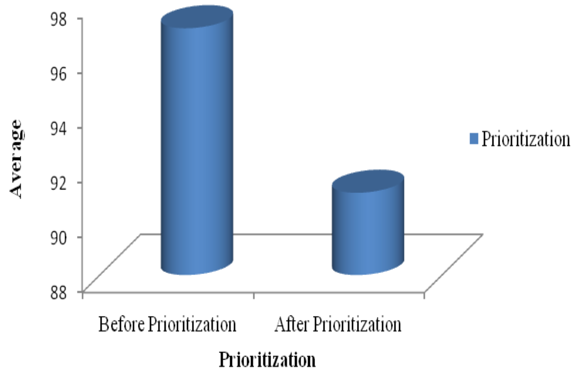

**Figure 8:** Graph for Before and After prioritization

*B. Comparative Analysis*

The Existing review works are compared in this section with the proposed work to show that our proposed work is better than the state-of-art works. We can establish that our proposed work helps to attain very good accuracy for the estimation of software quality database using Improved cuckoo search algorithm. And also we can establish this preoposed accuracy outcome by comparing other existing work. We have utilize Hybrid RTS prioritization algorithm for our Comparison in our work. The Comparison outcomes are presented in the following table III.

**Table VII:** Result of the Comparison of Proposed vs Existing

| Method | Before Prioritization | After Prioritization |
|---|---|---|
| Proposed Method | 97% | 91% |
| Existing Method | 70% | 80% |

Below specified fig.9 explains the Comparison of the Proposed vs Existing method.

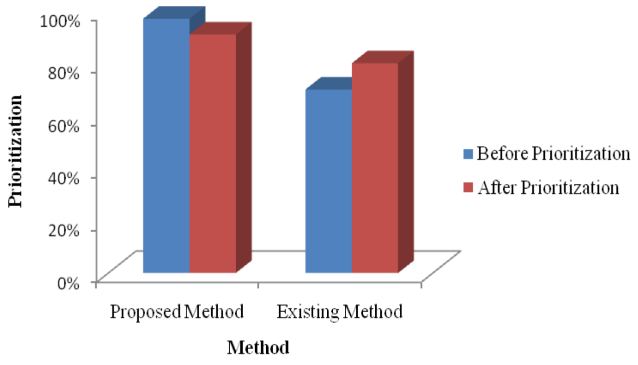

**Figure 9:** Graph for Comparison of Proposed vs Existing method

The improved good accuracy outcomes of our proposed work. In comparison with the Hybrid RTS prioritization algorithm gives very less accuracy values for the before and after prioritization. The before proritization of the existing method for the Hybrid RTS prioritization algorithm are 70% which is low in compared with our Proposed method 97%. The after prioritization of the existing method gives 80% which is low in compared with our proposed method 91%. From these outcomes, it is known that by means of our study provides very good for the software quality estimation purpose as it gives improved accuracy outcomes. In our proposed study introduced this can be a software quality estimation centered on PSO, Improved Cuckoo search algorithm and Prioritization. Hence the efficiency methods computation revealed which our Proposed method is successful than the Existing method.

## CONCLUSION

An Improved Cuckoo search based software quality estimation with four phases, they are Feature extraction, optimization using prioritization, optimization using improved cuckoo search, was proposed in this paper. The testcases are produced from the application program after the features are extracted from the testcases then by utilizing the PSO algorithm to optimize the features. Then these features are optimized by using improved cuckoo search algorithm and presented features quantitatively, and during the planning and execution of software development. From the outcomes, we have showed that the our improved cuckoo search optimization algorithm utilize in our proposed work outperforms the other optimizationtechniques as hybrid RTS algorithm by facilitates very good accuracy. Thus, we can observe that our proposed work is better than other existing works for the software quality estimation. Our planned process has accomplished a prioritization value in after and befor prioritization we got 91% and 97%.

## REFERENCE

[1] Matt Staats, Pablo Loyola, Gregg Rothermel, “Oracle-Centric Test Case Prioritization”, In proceeding of IEEE International Symposium on Software Reliability Engineering, pp.311-320,2012.

[2] E. Ashraf, A. Rauf, and K. Mahmood, “Value based Regression Test Case Prioritization”, Proceedings of the World Congress on Engineering and Computer Science, vol.1, 2012.

[3] Prakash N, Rangaswamy T.R, “Weighted Method For Coverage Based Test Case Prioritization”, Journal of Theoretical and Applied Information Technology, vol.56,no.2,pp.235-243,2013.

[4] Kun Wu, Chunrong Fang, Zhenyu Chen, and Zhihong Zhao, “Test case prioritization incorporation ordered sequence of program elements”, In proceeding of IEEE International Workshop on Automation of Software Test, pp.124-130, 2012.

[5] ZHANG Zhi-hua MU Yong-min TIAN Ying-a, “Test Case Prioritization for Regression Testing Based on Function Call Path”, IEEE International Conference on

Computational and Information Sciences, pp.1372-1375, 2012.

[6] Amr AbdelFatah Ahmed, Dr. Mohamed Shaheen, and Dr. Essam Kosba, “Software Testing Suite Prioritization Using Multi criteria Fitness Function”, IEEE International Conference on Computer Theory and Applications,pp.160-166, 2012.

[7] Daniel Di Nardo, Nadia Alshahwan, Lionel Briand, AND Yvan Labiche, “Coverage Based Test Case Prioritization: An Industrial Case Study”, In proceeding of IEEE International Conference on Software Testing, Verification and Validation,pp.302-311,2013.

[8] Md. Junaid Arafeen and Hyunsook Do, “Test Case Prioritization Using Requirements-Based Clustering”, In proceeding of IEEE International Conference on Software Testing, Verification and Validation, pp.312-321,2013.

[9] S. Raju, and G. V. Uma, “Factors Oriented Test Case Prioritization Technique in Regression Testing using Genetic Algorithm”, European Journal of Scientific Research, vol.74, no.3, pp.389-402, 2012.

[10] P.Bharath Kumar C.Lakshminatha Reddy, and V.Surendra Gupta, “Creating a Test Case Prioritization Technique Using Reliance Estimation of Functional Requirement”, IEEE International Journal of Engineering Trends and Technology, vol.18, no.2, pp.88-92, 2014.

[11] Albert Pravin and Subramaniam Srinivasan , “Effective Test Case Selection And Prioritization in Regression Testing”, Journal of Computer Science, vol.9, no.5, pp.654-659, 2013.

[12] Nitika Sharma and Neha Malhotra, “Regression Testing Prioritization, Selection and Reduction using Hybrid Criteria”, In proceeding of IEEE International Journal of Computer Applications,vol.95,no.7,pp.38-46,2014.

[13] Neha Sethi, Shaveta Rani, and Paramjeet Singh, “Ants Optimization for Minimal Test Case Selection and Prioritization as to Reduce the Cost of Regression Testing”, IEEE International Journal of Computer Applications, vol.100,no.17,pp.48-54,2014.

[14] Bansal, Priyanka. "A critical review on test case prioritization and Optimization using soft computing techniques."In proceeding of IEEE International Conference on Role of Technology in Nation Building, pp.74-77, 2013.

[15] Pravin, A., and S. Srinivasan. "An efficient algorithm for reducing the test cases which is used for performing regression testing." Proceedings of IEEE International Conference on Computational Techniques and Artificial Intelligence, pp.194-197, 2013.

[16] Hong Mei, Dan Hao, Lingming Zhang,Lu Zhang, Ji Zhou, and Gregg Rothermel, “A Static Approach to Prioritizing JUnit Test Cases”, IEEE transactions on software engineering, vol. 38, no. 6, pp.1258-1275,2012.

[17] Bestoun S. Ahmed, Mouayad A. Sahib, Moayad Y. Potrus, “Generating combinatorial test cases using Simplified Swarm Optimization (SSO) algorithm for automated GUI functional testing”, Engineering Science and Technology, an International Journal,vol.17, pp.218-226, 2014.

[18] Ke Zhai, Bo Jiang, and W.K. Chan, “Prioritizing Test Cases for Regression Testing of Location-Based Services: Metrics, Techniques, and Case Study”, IEEE transactions on services computing, vol., 7no.1, pp.54-67, 2014.

[19] Sreedevi Sampath, Rene ´e Bryce and Atif M. Memon, “A Uniform Representation of Hybrid Criteria for Regression Testing”, IEEE transactions on software engineering,vol.39,no.10,pp.1326-1344,2013.

[20] Ashima Singh “Prioritizing Test Cases in Regression testing using Fault Based Analysis”, IEEE International Journal of Computer Science Issues, Vol. 9, NO.6, PP.414-420, 2012.

[21] Ahlam Shakeel Ahmed Ansari, Prof. K. K. Devadkar, Dr. Prachi Gharpure, “An optimized technique for test suite refinement in regression test”, IEEE International Journal of Advancements in Research & Technology, vol.2, no.7,pp.263-266,2013.

[22] Gurinder Singh, Dinesh Gupta “An Integrated Approach to Test Suite Selection Using ACO and Genetic Algorithm”, International Journal of Advanced Research in Computer Science and Software Engineering vol.3, no.6,pp.1770-1778,2013.

## AUTHOR BIOGRAPHY

**K. Senthil Kumar** obtained his Bachelor’s degree in Computer Science from University of Madras in 1994. Then he obtained his Master’s degree in Computer Applications. Currently, he is the Assistant Professor at MCA Department, SRM University, Chennai. His specializations includes Software Testing and Software Engineering. His current research interest is Test Case Prioritization in Software Testing.

**Dr. A. Muthukumaravel,** He is Professor & Head, Department of MCA, Also director in faculty of Arts & Science, Bharath University, Chennai. He has completed his Phd in Computer Science from Vels University, Chennai in 2012. His research interest areas are Artificial Neural Network, Fuzzy Logic, Software Engineering. He is also a board member in International Journal of Computer Trends and Technology (IJCTT)