Test Case Generation Process using Soft Computing Techniques

Baswaraju Swathi, Harshvardhan Tiwari

Abstract-Software testing is the SDLC's important and most expensive step. Software testing is difficult and time-consuming work requiring a great deal of money for software development. Testing is both an operation that is static and adaptive. Software testing process deals with the creation of test cases, checking and validating either passed or failed test cases. It is unidealistic to check only the discerning parts of the material as a whole at once. It is not possible to test the whole system once, so selected parts of the code are considered for analysis. Since the input space of the Product Under Test (PUT) can be very large, it is important to analyze a representative subset of test cases. During software testing, the most important task is to build appropriate test cases. An effective set of test cases can detect more errors. Software testing always requires high deficiencies. Test cases are constructed using the test data. In the automation of software testing, the important task is to generate test data according to a given level of competence. The improved test data are determined using the test case development methodology and the test data adequacy criterion being applied. For increase the level of automation and performance, these aspects of test case development need to be studied. This paper studies the various test case generation techniques using soft computing techniques like Genetic Algorithm, Artificial Bee colony methods. Further an evaluation criterion for the test case generation process, empirical study of Code Coverage and its importance is discussed.

Keywords- Program Under Test, Test case Generation, Test data, Soft computing, Genetic Algorithm, Artificial Bee Colony

I. INTRODUCTION

Testing is generally a process rather than a single operation. This process starts with the scheduling of tests, the layout of test cases, the preparation for implementation and the evaluation of the condition until the end of the test. The following basic steps categorize the tasks in the evaluation cycle: Setup phase, Analysis and Design phase, Execution phase, Evaluation phase, Closure Phase, test case generation process is the most significant development in software testing [1],[2],[3],[4],[5]. Soft computing techniques provides a set of procedures, that recommend defect, deception, uncertainty and unfinished truth to accomplish tractability, strength, and low arrangement cost for a system. Soft computing methods can be used to predict fault liability of software using previous statistics. The methods use Genetic Algorithms, Adaptive network-based fuzzy inference, Support vector machine and artificial neural networks. Inadequate infrastructure and the necessity for rapid deployments have led Soft Computing techniques to be adapted to software testing. The aspects of testing-manual testing, user acceptance testing, and testing in real-world client environments to get a broad view about the quality of the product. 80% of the testing is the duplication of identical tests of features the software had earlier (Regression Testing), the difficult part is spending human resource and cost for manual testing for a work that could be done by Soft Computing. It has been observed that Test suite optimization, Log Analytics, Traceability, Customer sentiment Analytics, Defect analysis are the main aspects of testing which soft computing techniques can be applied.

Revised Manuscript Received on November 06, 2019.

***** Correspondence Author
Baswaraju Swathi*, Research Scholar,CSE, Jain University Bangalore ,Karnataka,India.Email: baswarajuswathi@gmail.com
Dr.Harshvardhan Tiwari, Associate Professor, Centre for Incubation, Innovation, Research and Consultancy,Jyothy Institute of Technology,Bengaluru,Karnataka, India.
Email: tiwari.harshvardhan@gmail.com

II. GENETIC ALGORITHM

A Genetic Algorithm is a tool based on a common selection tradition that simulates biological evolution to solve constrained and unconstrained optimization problems. The algorithm updates a population of individual solutions over and over again. The genetic algorithm randomly chooses individuals from the current population at each step and uses them as parents to make the children for the next step.

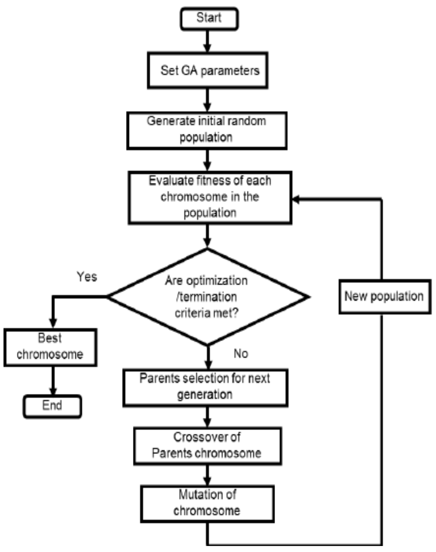

Start
Set GA parameters
Generate initial random population
Evaluate fitness of each chromosome in the population
Are optimization /termination criteria met?
Yes
No
Best chromosome
End
New population
Parents selection for next generation
Crossover of Parents chromosome
Mutation of chromosome

Fig.1. Genetic Algorithm

Genetic Algorithm in Software Testing [1], Test Cases and Test Data Generation are the main software testing activities and their computerization increases competency and efficiency, minimizing software testing costs.

The generation of test cases from the systemic method to testing through the analysis of branch and statement coverage, branch testing and genetic algorithm route testing.

III. OVERVIEW OF THE TEST CASE GENERATION TECHNIQUES

The test case development process deals with the design of test cases, the preparation of test data, the running of test data system and the analysis of results, the scope of tests.

A. Specification-based

These techniques[5] are methods from specification manuals to generate a series of test cases, such as a structured specification file. Specification documents clarify the functionality of the process. The advantages of this approach include the ability to use the specification document to obtain approved test data results, and tests can be produced concurrently with design and implementation. The design test generation process will make it easier for the software engineer to assess configuration issues. Early elimination of the problems can save time and resources if the tests are created early. The generation of test cases may be independent of the specification. Tran[6] focused on existing study in to generate test cases through model checking. Several areas have been measured, such as the software testing method, the use of model checking to generate test suits and the specification for test generation needs. Specifically, this approach relies on mutation analysis to produce test cases, a white-box technique to create a collection of test cases that are resistant to any minute syntactic changes to a program's structure. Rayadurgam and Heimdahl[7] have suggested a method for generating test cases automatically based on criteria for structural coverage.

Table-I: Requirement Type and Testing Level applicable

Requirement Type	Testing Level
Functional Requirement	Acceptance, System/Integration, Uni t testing
Non-Functional Requirement	Acceptance, System/Integration, Uni t testing

B. Sketch Diagram-based

Sketch-based methods are used to create test cases from diagrams such as use case diagrams, system diagrams. Ryser and Glinz[8] suggested an approach that discusses software testing documentation, scheduling and hazard concerns, scenarios / UML use cases, and system diagrams to derive test cases. When researching web-based applications, Nilawar and Dascalu[9] used sketch-based techniques. Web-based applications are of increasing density and it is a solemn business to test them accurately, focusing on black box testing that allows sets of input conditions to be derived that will fully apply the functional requirements. The Planned method is a four-step process, prioritize use cases based on the requirement traceability matrix, generate adequate use cases and test scenarios, identify at least one test case for each scenario, and identify test data values

for each test case. Sinha and Smidts[10] have specified a new model-based test technique that has been developed to classify significant domain needs. Challenges: With limited time and resources end up with inefficient test cases are generated. Critical Domain requirements may be missed.

Table- II: Requirement Type and Testing Level applicable

Requirement Specification Technique	Testing Level	Testing Scope
Use cases, State charts, Natural languages, Formal Specification languages	Acceptance, System/Integration, Unit testing	White box Testing, Black box testing

C. Source Code-based

Source code-based techniques [11] generally use control flow information to create a set of paths to cover and to generate suitable test cases for these paths. It is possible to derive the control flow graph and the optimized control flow graph directly from source code. A novel approach to automated test case development was proposed by Beydeda and Gruhn[12].Number of approaches for test case generation have been predicted, mainly random, source-based, goal-oriented and smart approaches (Pargas et al., 1999). Random methods analyze test cases based on fault distribution assumptions. Turner et al.[13], suggested an activity-oriented approach to test web applications; it is a web application-based black box analysis based on user experiences. When web applications become more complex, with connections, buttons and multiple forms, the design of web pages has become difficult to understand, challenging and overloaded. While predictable, manual testing of such web applications is not effective. The design of automated tests that can reflect deficiencies and deviations from the proposed behavior is therefore necessary. User experiences can be as simple as clicking a button and complex as filling out multiple forms to achieve a mission. Measurement of these user interactions; test design can be extended to functional testing and load testing. Considering the Triangle Classification' program (Lin and Yeh, 2000; Ahmed and Hermadi, 2008), Given three sides of triangle, the program checks whether the three sides form a triangle, if so the type of triangle equilateral, scalene, isosceles are classified. The program is a non-loop program with stacked if-else of level 3.The OCFG for the program and the initial population from the Test Suit can be constructed as:

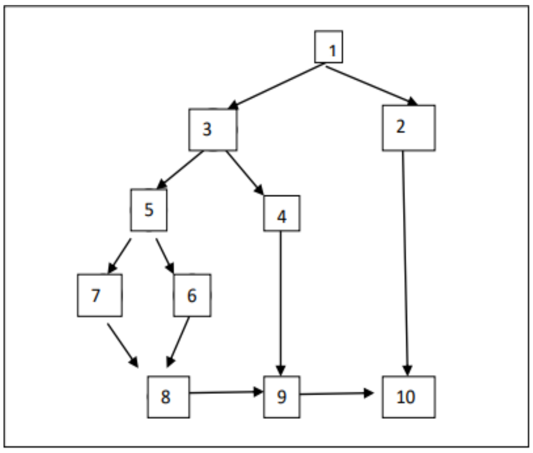


Fig. 2. Control Flow Graph for Triangle

The above OCFG satisfies all the definitions of

Cyclomatic Complexity. C=E-N+2: 12-10+2=4
C=P+1=3+1=4 (Predicates are Boolean conditions in Algorithm)
C=R+1=3+1=4(Internal Region+1 External Region) ALL Path Coverage generates the following paths:
1-2-5-7-8-9-10
1-2-10
and so on…, these paths are further considered as test cases.

D. Test Case Generation Method

This paper[14] aims to improve an automated method of test case creation to reduce a number of test cases while enhancing the ability to identify specific requirements for important domains. It introduces a new test case generation system with a requirement priority approach for overcoming relevant constraints for inefficient test case generation techniques, critical areas. Proposes a method of 2D-5A-4D prioritization test case development to solve problems in the generation of test cases.

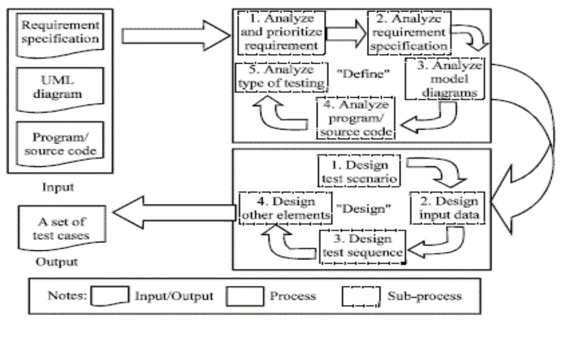


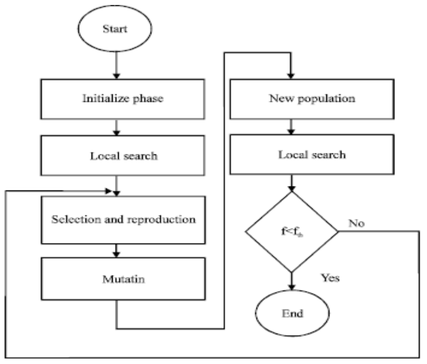
Fig.3 -2D-5A-4D

The above classifies the requirements as critical domain requirements or non critical domain requirements based on MoSCoW method, requirement prioritization using cost valued approach. The functional requirements are assigned with high priority, then implementation cost using cost value approach is calculated. Total cost is calculated as (WValue1*CostImp)+(WValue2*CostTest),Weight value for each requirement is calculated as (TC)*(Imp) where TC is considered as total cost. The three metrics: number of test cases, total time and percentage of the specification scope of critical domain.

E. Generation of automated computer test case

This paper[15] analyzes the application of various machine learning algorithms for automated test case creation, specifically for white box research.GA, MA and GA-NN algorithms are applied as learning algorithms for test case generation. The general factors considered for Genetic Algorithm are: Population Initialization, Fitness function, Likelihood, Close to boundary value, Branch Coverage, Selection/Operators used, Termination Criteria. Initial populations of test cases are considered to be random. Fitness function used as the average of three factors which are Likelihood, the paths which are more likely to be executed than other paths and given as where t1)is the path

followed when t1 is executed, is the likelihood of the path .Boundary close value is the other factor, B is a test suit, B(T)=B(t1)*B(t2)…Branch Coverage indicates percentage number of edges of the control flow graph, evaluated as P(T)= v/€.This system uses the features of efficient test suits and provide a high fitness value to generate test near to efficient ones.



Schema of the memetic algorithm. f_{th} denotes a fixed threshold value used for the stopping criterion

Fig.4. Schema of the mimetic algorithm

F. Optimization -UML Models and GA

This paper[16] proposes a method for generating test cases by combining UML sequence diagram with state diagram representing the code specifications applicable to a system, this approach converts sequence diagram into sequence diagram and converts state diagram into state diagram map. Through combining the two graphs, the System testing graph (SYTG) is created. Sequences of control flow are known as test cases and then optimized using evolutionary algorithms such as the Genetic Algorithm. State diagram reveals unit level flaws and sequence diagrams reveal integration level flaws, the combined test cases are considered suitable for testing the system as a whole, applying Genetic Algorithm to the created test suite optimizes the outcome. The system is labeled an online voting system, the state chart diagram graph is formed from the state diagram and the sequence diagram graph is formed from the sequence diagram. The SDG represents four major situations, followed by the incorporation of SG and SCDG into SYTG. SYTG generation considers SCDG and SG to be the inputs and SYTG to be the predicted output. With the following initialization of genetic algorithms, the genetic algorithm is used to produce optimized test cases: Population: All the paths of SYTG, Fitness value: F(x)=x*x, cost(x) is calculated on basis of weight assigned to each node. Probability is calculated as F(x)/sum(x), Crossover applied is single point intersect on 4th bit from right side, mutation is applied on every 4th bit only where random number <0.2.The above was applied on online voting system and the optimal path was generated to be 4th path with cost 153.

G. Generation of automated computer test case- mutation analysis

The proposed method [17], considers introducing the mutant in the program that is change the statements in the source code and test for the bugs. The system later generates random test cases, computes the mutation score and if it is acceptable it terminates otherwise refinement of the test cases is through mutation score. Mutation testing is considered to provide high coverage, and the Genetic Algorithm is designed to optimize the test cases. The suggested algorithm: the program injects mutants produce random test cases, the mutation score is calculated as the number of mutants / total number of mutants, test cases with a mutation score of less than 20 percent are not further considered. Test cases with greater than 20% are considered and Genetic algorithm is applied on this population to obtain optimized test cases.GA factors considered in this system are: Population: Test cases with mutation score >20%,Selection process uses tournament selection, Mutation and crossover applied greater than 50% of mutation score achieved by the test cases.

H. Hybrid GA

This paper[18] proposes generating test cases based on sequence diagrams and applying Genetic Algorithm. The model proposes method for sequence diagram creation using genetic algorithm to investigate process sequences primarily to produce functional actions in the application domain. Method sequences obtained through this system are used to create dynamic execution test cases. The developed test cases are intended for testing the level of integration. The architecture proposed accepts the sequence diagram as input and generates as output a series of test cases. The sequence diagram is used here for the efficient flow of events, which provides a full one-use scenario for the given application as the basis for the generation of test cases.GA is used to search call sequences which gives alternate paths. Chromosome generator generates chromosomes from sequence diagram. Encoding Gene deals with Constructor Call Gene, Simple Method invocation gene and object value assignment gene. Objective function is designed in a way that it depends on coverage criteria for testing, fitness function is considered to make the most of the coverage. Experimental Analysis on stack, calculator, and Course Enrolment system were conducted generated test cases covering full message sequence.

Table –III: Empirical Evaluation

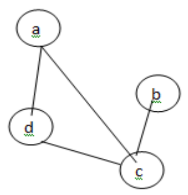
Program considered	No of methods	No of methods covered using above proposed work
Stack	9	8
Calculator	5	5
SCES	30	22

I. All- Edge Coverage Criteria

This paper[19] proposes a GA method to detect the test program's sub-set of paths that satisfy all edge coverage requirements. All edge criteria usually covers all the possible test cases in a source program. The (GATCG) designed in this work creates a compressed number of paths under review for a test program, generating prime paths. The GATCG methodology suggested uses the idea of primary paths to reduce testing costs. The aim is to find a subset of target paths that sufficiently represents the complete set of all paths. GATCG algorithm based on the concept of genetic algorithm was proposed in this paper to produce tests that provide good coverage for all-edge test criteria. The GATCG technique based on Genetic Algorithm proposed is an effective and efficient method of generating test cases. The fitness function used evaluates the chromosomes and promotes the chromosomes with high percentage of connected paths, to pass on to the next generation. The proposed approach derives a Control Flow Graph and optimizes the graph.GA factors are considered as follows: Population: Set of paths from {P1,P2…Pn}, A path is considered as flow from start node to end node of a OCFG. Fitness Function is calculated as: FFN(Ci)=Total number of nodes covered by Ci+1/Number of total nodes present in Ci. Selection- Based on their fitness values, the choice of parent chromosomes is suggested. The algorithm uses the roulette wheel selection method to select test paths from current population members once the fitness is calculated in the current population. Crossover: Crossover operates at the chromosome level with a programmed probability CP. A single point crossover is considered to produce off springs. Mutation operates subsequent to crossover operator, works at cell level. In mutation each field is altered with encoded mutation probability MP. If the field value is 1 and the node signifying that point has sibling, present field value is initialized to zero and its sibling’s field value is made binary 1 irrespective of sibling’s preceding value. Field value of nodes with no sibling(s) is not manipulated

J. Genetic Algorithm and Graph Theory

This paper[20] proposes an method to discover the optimal minimization of the untested paths in the program and to enhance the coverage criteria. The method determines the automata of the state transition based on graph theory concepts. A directed graph is generated and the complete nodes are considered to be the initial population, further the GA operations such as crossover, mutation are applied on this data. For generating the next level population the complete graph is covered at least once, new population is obtained.
The suggested architecture assumes that the
primary data structure used in GA is bit strings, providing a way to represent the graph using the matrix of incidence.



$$\begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 0 & 1 & 1 \\ b & 0 & 0 & 1 & 0 \\ c & 1 & 1 & 0 & 1 \\ d & 1 & 0 & 1 & 0 \end{array}$$

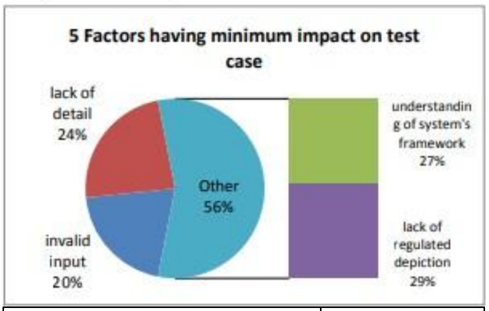
0 1 1 1 0 1 is the bit string.

Fig 4 Bit String of Graph

GA variables used here are: Population: the search space considered here is the random sample of various size 43 graphs that are independent of running. Crossover action uses chance based on fitness. Summation (f(sj(t)),Sj(t) is the cumulative summation of population members, copied to the next generation, the second method uses tournament choice, GA randomly selects two solutions, solution with higher fitness is considered. The third method here considers Rank based selection. Mutation operation considered here is of two kinds, first a function replacing a function and a terminal replacing a terminal, second entire sub tree replaces another sub tree. Steps followed are converting the original graph to a dual graph, Eulerizing the dual graph, generated sequence is considered to be the initial population. Further the GA operations are applied.

K. TC-Genetic Algorithm

In view of the factors that touch level on test case, this paper[21] proposes an approach to generates t cases through GA. This includes the aspect level, system's structure, test design techniques, precise suppositions, test case evaluation. The self-possessed methodology is satisfactory to the relations configuration measures and provides a even flow of statistics opening with one stage then onto the next. The main point is to assess the possibility of using GA to construct upgraded test information for programming testing.



4 Factors	Impact level
Mis-usage of techniques	80%
Tidy up after execution	78%
Lack of Variation	75%
Unrealistic supposition	70%

Fig 5. Factors having High-Impact

L. Test Case generation considering Improved Genetic Algorithm

This paper[22] proposes an improved version of Genetic Algorithm which aims to increase the search speed and rate of convergence by means of binary encoding, the selection operation of GA uses Roulette wheel selection, random cross over function and fitness value calculated as per the requirements. Results were compared and resulted in considerable enhanced. The proposed method focuses mainly on binary encoding process with minimum character set encoding theory, which is proposed separately for multiple parameters encoding binary bit strings.
F=(100/(1+f1)+100/(1+f2)+100/(1+f3))/3, where
f1, f2, f3 are the system branch functions being evaluated, triangle classification is taken into account. Experimental study on Triangle classification problem has proven advantages compared to Basic GA over Improved GA.

Table-IV: Empirical Results

Methods considered	Normal Running Algebra(G)	Normal Running Time/s(T)
Basic Genetic Algorithm	112	3.5
Enhanced GA	23	1.3

M. Particle Swarm Optimization

Particle swarm Optimization[23] is a population-based optimization technique inspired by social behavior is adapted in this paper, test case generation based on SAF-GPSO strategy is used The multiplicity of population as a cluster operation is a quantitative survey of swarm activity carried out by population diversity operators. The group action is reduced; the modified process of Gauss inertia weight generates the new weight of inertia. Phase 1: Initializing the swarm of particles. Step 2: Using the fitness function, calculate the current particle position value generation, modernize the optimal local and global position. Step 3: figure out whether the current iteration number reaches the limit or whether the fitness variable is 0.Step 4: Iterations add 1 and update the test case hustle and arrangement according to the equation.

$$v_{id}^{(t+1)} = wv_{id}^{t} + c_1r_1(p_{id}^{t} - x_{id}^{t}) + c_2r_2(g_{id}^{t} - x_{id}^{t})$$

$$x_{id}^{(t+1)} = v_{id}^{(t+1)} + x_{id}^{t}$$

Step5:Calculate the usual particle range by (4); Step 6: When group operation is compressed, Gauss inertia weight (5) produces a novel inertia weight and then use the recently obtained inertia weight except falling executes step 2; Phase 7: Performance shows the best position possible.

N. Many-Objective Optimization

This paper[24] introduces DynaMOSA (Dynamic Many-Objective Sorting Algorithm), a new multi-objective solver intended primarily to focus on the question of test case creation in terms of coverage analysis. The Algorithm improves the former MOSA (Many-Objective Sorting Algorithm) multi-objective technique by actively selecting the coverage objectives based on the command dependency hierarchy; the approach is efficient and competent in the event of an Inequitable budget. Experimental analysis has been given and better results have been deferred than MOSA. The paper offers in-depth analysis of many practical methods for the generation of test cases, an experimental study was conducted on 346 java classes sampled from various data sets. DynaMOSA-a new algorithm for dynamic target selection, technically verified as subsuming MOSA deals with larger-scale experiments:- the new experiments find one order of magnitude of additional classes. DynaMOSA experimental analysis with MOSA and WSA is applied to provide guidance for the use of other analytical solvers. Different check parameters, branch coverage, claim coverage and mutation coverage are called statistical evaluation of certain algorithms w.r.t. This leads to the contrast with Pareto's dominance-based algorithms, which is NSGA-II.Empirical evaluation of preferential criteria is evaluated to determine whether DynaMOSA's higher efficiency is due to the preference criterion or its dominance ranking permutation.

O. Multi-Objective Search

This paper[25] introduces a multi-objective approach for measuring the industrial cyber physical system by defining a fitness feature with four targets, in addition to five multi-objective search algorithms (e.g., Undominated Genetic Algorithm Sorting (NSGA-II)), Development and experimental evaluation of the fitness function and the expected operators. Results were compared with previous methods and significant improvements were made, NSGA-II advancing Random Search by 43.80% on average for each goal and 49.25% for the Hyper Volume (HV) value indicator. The proposed method considers scope of criteria, similarity of test cases and similarity of test cases based on priority, time of execution of tests as cost-effective measures. Representation of the solution is a test suite consisting of at least one test case, each test case consisting of TCi={S1,S2,S3 ... Sn}.Cross-over operation exchanges the test cases between the two solutions, considering that two test suites initially select the cross-over point randomly.

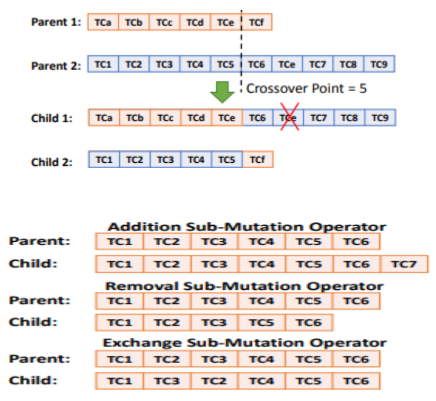

Parent 1:
TCa TCb TCc TCd TCe TCf
Parent 2:
TC1 TC2 TC3 TC4 TC5 TC6 TCe TC7 TC8 TC9
Crossover Point = 5
Child 1:
TCa TCb TCc TCd TCe TC6 TCe TC7 TC8 TC9
Child 2:
TC1 TC2 TC3 TC4 TC5 TCf
Addition Sub-Mutation Operator
Parent: TC1 TC2 TC3 TC4 TC5 TC6
Child: TC1 TC2 TC3 TC4 TC5 TC6 TC7
Removal Sub-Mutation Operator
Parent: TC1 TC2 TC3 TC4 TC5 TC6
Child: TC1 TC2 TC3 TC5 TC6
Exchange Sub-Mutation Operator
Parent: TC1 TC2 TC3 TC4 TC5 TC6
Child: TC1 TC3 TC2 TC4 TC5 TC6

Fig 6 Mutation operations derived from test suites

Experimental study results proved NSGA-II along with the fitness function; operators demonstrated the finest performance and outperformed the remaining algorithms on average between 11.93% and 47.07% [HV quality indicator]. To obtain the finest efficiency, the crossover rate of 0.2, 2/N mutation rate is suggested.

P. String Test case Generation

The paper[26] proposes an multi objective approach to black box string test case generation. Wide variety of test cases is considered to be the first objective measured by string distance calculation and the second objective function is considered as string length distribution, applying two objective functions enhances string tests.GA proposed here considers the diversity-based fitness variable, dist is the distance from each test case to its nearest test case.

GA parameters: initial population is a string test set randomly generated string test is considered, selection mechanisms considered here is a rank based selection which produced better results, Test cases are crossover to generate off-spring test sets at a rate of 60%, test sets are recombined, each string of the first parent test set is combined with the resulting string in the second parent test set and two string kids are generated, repeated for all string test cases in the parent test sets resulting in two offspring test sets. Single point crossover is applied on the test sets. Multi objective GA is proposed with string length fitness function, For both fitness functions at the same time, a multi-objective optimization is applied. To generate the test cases as per the NSGA-II. 1) Random initial population is generated. 2) Initial population is sorted.3) The next gene population of size N is formed using selection, crossover and mutation mechanisms.4) The current population of size 2N is formed.5)New population is sorted and the initial N chromosomes are selected for the next generation. 6) The criteria are stopped. Return to step 3 if it fails to meet the criterion. A Pareto-optimal set of test sets is produced by NSGA-II as a single optimal test set.

Q. Reinforcement Learning

The paper[27] proposes a prototype EvoQ based on isomorphism substitution actions which assist the evolutionary techniques, resulted in higher branch coverage mainly intended to generate Inherited Class Hierarchies and Non-public methods.

R. Knowledge Discovery Meta model-based

Unit level testing is language dependant. This paper [28] proposes an approach, using knowledge discovery to create a common representation which performs unit testing irrespective of platforms. The common format is considered to be an xml file described as KDM Meta model further generates test cases in generic Xunit Complaint.

S. ABC for T-way testing

The use of t-way testing strategies in testing is to construct a best possibility test case. Research is been done in various problems solved using T-way testing but as it is considered to be NP hard problem its not widely accepted, the proposed work[29] here combines Artificial Bee colony and Pair wise Analysis to generate the test cases. ABC is commonly used problem of optimization. The system proposed here considers a hypothetical system with exhaustive test cases, the major parameter called interaction strength is decided as 2-way testing. The population is initialized as possible solutions by Xij=Xjmin+rand(0,1).(Xjmax-Xjmin),new route is created using Vij=Xij+Øij.(Xij-Xkj).where minimum possible solutions and total solutions, maximum solutions are defines respectively. Ø defines random real number, rand defines random value between 0 and 1.

Fiteness function considered here is

$$fitness\ (x_i)\begin{cases}\frac{1}{(1+f(x_i))}, f(x_i) \geq 0\\ 1+|f(x_i)|\ , f(x_i) < 0\end{cases}$$

$$F_D = \sum_{i=1}^{test\ set\ size} dist(t_i, \beta(t_i, test\ set))$$

IV. TEST PARAMETER-CODE COVERAGE AS AN EVALUATION CRITERIA

Code coverage[30] defines the extent of execution of a program's origin code while conducting a specific test case / test suit.A program with high code coverage, measured as a percentage, ha a lower chance of undetected bugs. Test case generation are the techniques for automatic generation of test cases which will attempt to discover a miniature set of cases that permit an competence criterion to be satisfied, reducing the cost of software testing and ensuing in efficient testing of software products/systems. Considering the Triangle Classification' program (Lin and Yeh, 2000; Ahmed and Hermadi, 2008),Given three sides of triangle, the program checks whether the three sides form a triangle, if so the type of triangle equilateral, scalene ,isosceles are classified. The program is a non-loop program with stacked if-else of level 3.

```
void Triangle_Classification (int sidea , int sideb, int sidec)
Begin 1 if(
(sidea+sideb<=sidec)||(sidea+sidec<=sideb)||(sideb
+sidec<=sidea))
  2           System.out.println(”Not a triangle”); else{
3             if( (sidea==sideb))&&( sideb==sidec))
  4           System.out.println (” Equilateral Triangle”; else
  5            if((sidea==sideb)||( sideb==sidec)||(
sidea==sidec))
 6            System.out.println(“Isosceles Triangle”) else
7             System.out.println (“Scalene Triangle”)
 }}}
8              End
```

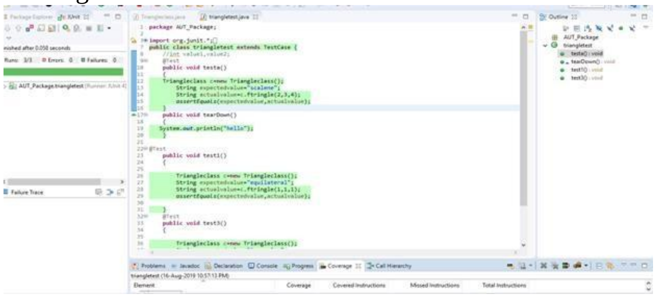
Consider the program to test the above Triangle Classification Program,

```
import org.junit.*; import org.junit.Test;
import junit.framework.TestCase;
public class triangletest extends TestCase { @Test
public void testa()
{
    assertEquals(expectedvalue,actualvalu e);
}
}
```

The following Experiment was conducted to implement a code coverage tool w.r.t program under test. The test program on the Triangle Classification Algorithm has three test cases resulting in each type of a triangle, yet the result of the code coverage is observed as 65.6% on the Algorithm, on the other hand test program has 100% code

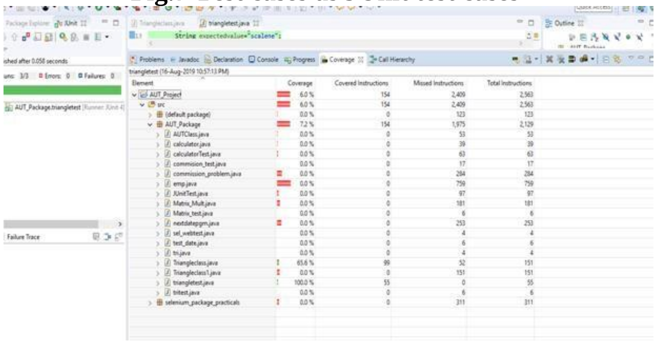
```
Triangleclass c=new Triangleclass(); String
expectedvalue="scalene"; String
actualvalue=c.ftringle(2,3,4);
assertEquals(expectedvalue,actualvalu e);
} @Test public void test1() { Triangleclass c=new
Triangleclass(); String expectedvalue="equilateral"; String
actualvalue=c.ftringle(1,1,1);
assertEquals(expectedvalue,actualvalu e); } @Test public
```

coverage.



```
package AUT_Package;

import org.junit.*;
public class triangletest extends TestCase {
    //int value1,value2;
    @Test
    public void teste()
    {
    Triangleclass c=new Triangleclass();
        String expectedvalue="scalene";
        String actualvalue=c.ftringle(2,3,4);
        assertEquals(expectedvalue,actualvalue);
    }
    public void tearDown()
    {
   System.out.println("hello");
    }

@Test
    public void test1()
    {

        Triangleclass c=new Triangleclass();
        String expectedvalue="equilateral";
        String actualvalue=c.ftringle(1,1,1);
        assertEquals(expectedvalue,actualvalue);

    }
    @Test
    public void test3()
    {

        Triangleclass c=new Triangleclass();
```

Fig.7 Test cases as JUnit test cases



triangletest (16-Aug-2019 10:57:13 PM)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
AUT_Project	6.0 %	154	2,409	2,563
src	6.0 %	154	2,409	2,563
(default package)	0.0 %	0	123	123
AUT_Package	7.2 %	154	1,975	2,129
AUTClass.java	0.0 %	0	53	53
calculator.java	0.0 %	0	39	39
calculatorTest.java	0.0 %	0	63	63
commision_test.java	0.0 %	0	17	17
commission_problem.java	0.0 %	0	284	284
emp.java	0.0 %	0	759	759
JUnitTest.java	0.0 %	0	97	97
Matrix_Mult.java	0.0 %	0	181	181
Matrix_test.java	0.0 %	0	6	6
nextdatepgm.java	0.0 %	0	253	253
sel_webtest.java	0.0 %	0	4	4
test_date.java	0.0 %	0	6	6
tri.java	0.0 %	0	4	4
Triangleclass.java	65.6 %	99	52	151
Triangleclass1.java	0.0 %	0	151	151
triangletest.java	100.0 %	55	0	55
tritest.java	0.0 %	0	6	6
selenium_package_practicals	0.0 %	0	311	311

Fig 8. Test coverage

V. SUMMARY OF TEST CASE GENERATION ALGORITHMS

Table-IV. Summary of Test Case Generation Algorithms

TestCase Generation Techniques	Advantages	Disadvantages
Specification Based	Attractive from budget perspective	1.Difficulty in formal analysis. 2.Great manual effort in generating test cases.
Sketch Based	Efficient test documentation with UML diagrams tool support formal language are used	1.With limited resources testcases generated are ineficient 2.Very few domain specifications. 3.size of testases is ignored.
Source Code based	Attractive for Unit Testers	1.Not much focus on other type of testing 2.Cannot assure meeting requirements as test cases are generated from source code
Soft Computing Techniques (GA)	Introduces appropriateness and effectiveness as heuristic algorithms are	1.Involves complex steps in the process. 2.Need for enhanced algorithms.

VI. CONCLUSION

This paper specifies the test case generation process and techniques mainly concerning soft computing. Though various Test case generation techniques exist, Test Case Generation has always scope of improvement.Traditonal ways of test case generation like source code based, specification based has their own limitations. Since test case generation can be considered to be an optimization problem Soft Computing Techniques can be applied. Detailed Analysis of the existing system was studied, based on which the evaluation criteria Test coverage is considered as an important parameter for measuring testing. Measuring Test coverage in JAVA Environment was studied as a case study for a triangle classifier problem.

REFERENCES

1. Sommerville, I., 2000. “Software Engineering”. 6th Edn., Addison-Wesley, England.
2. Bertolino, A., 2003. Software testing research and practice. Proceedings of the 10th International Workshop on Abstract State Machines, March 3-7, Taormina, Italy, 244-262.
3. Antonio, P.Salas and B.K. Aichernig, 2006. Automatic test case generation for OCL: A mutation approach, Proceedings of 5th International Conference Quality Software, January 2006, IEEE Computer Society, pp: 64-71.
4. Kaner, J.D.C., 2003.’ What is a Good Test Case’, Florida Institute of Technology, USA.
5. Jia,X.and H.Liu,2002,Rigorous and Automatic testing of web applications, Proceedings of 6th IASTED International Conference on Software Engineering and Applications, May 2002,Honolulu,USA
6. Tran H.2001,Test generation using model checking. European Conference on Software Maintenance and Reengineering,CSMR2001,http://www.cs.toronto.edu/~chechik/ courses00/csc2108/projects/4.pdf
7. Rayadurgam, S. and M.P.E. Heimdahl, 2001. Test-sequence generation from formal requirement models. Proceedings of the 6th IEEE International Symposium on High Assurance Systems Engineering, Oct. 22-24, Boca Raton, Florida, pp: 23-23.
8. Ryser, J. and M. Glinz, 2000. SCENT: A method employing scenarios to systematically derive test cases for system test. Technical Report, http://portal.acm.org/citation.cfm?id=901553.
9. Nilawar, M. and S. Dascalu, 2003. A UML-based approach for testing web applications. M.Sc. Thesis, University of Nevada, Reno.
10. Sinha, A., and C.S. Smidts, 2005. Domain specific test case generation using higher ordered typed languages from specification. Ph.D. Thesis, University of Maryland.
11. Jaya Srivastaval and Twinkle Dwivedi,SOFTWARE TESTING STRATEGY APPROACH ON SOURCE CODE APPLYING CONDITIONAL COVERAGE METHOD,International Journal of Software Engineering & Applications (IJSEA), Vol.6, No.3, May 2015
12. Beydeda, S. and V. Gruhn, 2003. BINTEST-Binary search-based test case generation. Proceedings of Computer Software and Applications Conference, November 2003, Leipzig Univ., Germany, pp: 28-33.
13. Turner, D.A., M. Park, J. Kim and J. Chae, 2008. An activity oriented approach for testing web applications. Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering, Sept. 15-19, Washington, USA .pp: 411-414.
14. Nicha Kosindrdecha and Jirapun Daengdej, A Test Case Generation Process and Technique, Journal of Software Engineering, 265-287, 2010.
15. Hajar Homayouni, Hossein Shirazi, Automatic Software Test Case Generation, Article in Journal of Software Engineering 5(3):91-101, March 2011 Academic Journals Inc.
16. Namita Khurana, RS Chillar, Test Case Generation and Optimisation using UML Models and Genetic Algorithm ICRTC-2015,Published by Elsevier
17. Rijwan Khan, Mohd. Amjad,2015 IEEE UP Section Conference on Electrical Computer and Electronics (UPCON) Automatic test case generation for unit software testing using genetic algorithm and mutation analysis
18. Mahesh Shirole, and Rajeev Kumar,A Hybrid Genetic Algorithm Based Test Case Generation Using Sequence Diagrams,International Conference on Contemporary Computing,IC3 2010:Contemporary Computing pp 53-63
19. Shveta Parnami,Krishna Swaroop Sharma,Empirical Validation of Test Case Generation based on All- Edge Coverage Criteria, International Journal of Computer Applications,September 2015.
20. Dr. Velur Rajappa, Arun Biradar, Satanik Panda, Efficient Software Test Case Generation Using Genetic Algorithm Based Graph Theory, First International Conference on Emerging Trends in Engineering and Technology,2008.
21. Ahmed Mateen, Marriam Nazir, Salman Afsar Awan,Optimization of Test Case Generation using Genetic Algorithm (GA),International Journal of Computer Applications (0975 – 8887),Volume 151 – No.7, October 2016.
22. Yuehua Dong, & Jidong Peng. (2011). Automatic generation of software test cases based on improved genetic algorithm. 2011 International Conference on Multimedia Technology. doi:10.1109/icmt.2011.6002999
23. Huang, M., Zhang, C., & Liang, X. (2014). Software test cases generation based on improved particle swarm optimization. Proceedings of 2nd International Conference on Information Technology and Electronic Commerce. doi:10.1109/icitec.2014.7105570.
24. Panichella, A., Kifetew, F. M., & Tonella, P. (2018). Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets. IEEE Transactions on Software Engineering, 44(2), 122–158.
25. Arrieta, A., Wang, S., Markiegi, U., Sagardui, G., & Etxeberria, L. (2018). Employing Multi-Objective Search to Enhance Reactive Test Case Generation and Prioritization for Testing Industrial Cyber-Physical Systems. IEEE Transactions on Industrial Informatics, 14(3), 1055–1066.
26. Shahbazi, A., & Miller, J. (2016). Black-Box String Test Case Generation through a Multi-Objective Optimization. IEEE Transactions on Software Engineering, 42(4), 361–378.
27. He, W., Zhao, R., & Zhu, Q. (2015). Integrating Evolutionary Testing with Reinforcement Learning for Automated Test Generation of Object-Oriented Software. Chinese Journal of Electronics, 24(1), 38–45.
28. Pires, J. P., & Brito e Abreu, F. (2018). Knowledge Discovery Metamodel-Based Unit Test Cases Generation. 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST).
29. Ali, M. S. A. R., Othman, R. R., Yahya, Z. R., Ahmad, M. Z. Z., & Ramli, N. (2016). Implementation of artificial bee colony algorithm for T-way testing. 2016 3rd International Conference on Electronic Design (ICED).
30. Khalid Alemerien and Kenneth Magel, Examining the Effectiveness of Testing Coverage Tools: An Empirical Study, International Journal of Software Engineering and Its Applications,2014.

AUTHORS PROFILE

Ms. Swathi B is a Senior Assistant Professor at New Horizon College of Engineering, Bengaluru. She has total experience of 9+ years in teaching, pursued Masters Degree in Software Engineering. Her research interests in the areas of: Software Engineering, software process improvement, software project management, Software Testing, object oriented technologies. She is currently pursuing Ph.D in Software Engineering and Soft Computing Techniques. She has authored 16 papers in international conferences and journals.

Dr.Harshvardhan Tiwari is currently working withCentre for Incubation, Innovation, Research and Consultancy(CIIRC), Jyothy Institute of Technology campus,Bengaluru, Karnataka, India in the capacity of an Associate Professor. Dr.Tiwari has received his PhD degree in Computer Science and Engineering from JIIT University,Noida,Uttar-Pradesh,India.He completed his Post Graduate study (MTech) and Graduate study (BE) both in Computer Science and Engineering. from RGTU,Bhopal, Madhya Pradesh,India in 2009 and 2005 respectively. His areas of interest includes Computer Networks, Algorithms, DBMS, Network security and Information Security,Cryptography, Information Security, Soft computing. He has more than 4 years of Teaching and Research experience. He has authored 10 International Journals and 3 International conference papers. Research papers are indexed in SCOPUS and DBLP. Dr.Tiwari has participated in various Faculty Development Programs and conducted value addition programs.