TEST CASE OPTIMIZATION USING NATURE INSPIRED ALGORITHM

Submitted by:

Vibhor Gupta	9913103515
Naman Sharma	9913103546
Avneet Singh	9913103676

Under the Supervision of

Mr. Himanshu Mittal
(Dept. of CSE/IT)

May - 2017

Submitted for the partial fulfilment of Degree of
Bachelor of Technology
Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING/IT

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY

(I)

TABLE OF CONTENTS

(II)

DECLARATION

We thus proclaim that this accommodation is my/our own particular work and that, to the best of my insight and conviction, it contains no material beforehand distributed or composed by someone else nor material which has been acknowledged for the honor of whatever other degree or recognition of the college or other establishment of higher learning, aside from where due affirmation has been made in the content.

Place: JIIT,NOIDA

Date: 13/05/2017

Signature:

Name: VIBHOR GUPTA

Enrollment No: 9913103515

Signature:

Name: NAMAN SHARMA

Enrollment No: 9913103546

Signature:

Name: AVNEET SINGH

Enrollment No: 9913103676

(III)

CERTIFICATE

This is to ensure that the work titled "**TEST CASE OPTIMIATION BY NATURE INSPIRED ALGORITHMS**" submitted by "**VIBHOR GUPTA, NAMAN SHARMA, AVNEET SINGH**" in satisfaction for the honor of level of B.Tech of Jaypee Institute of Information Technology University, Noida has been completed under my watch. This work has not been submitted incompletely or completely to some other University or Institute for the honor of this or some other degree or recognition.

Signature of Supervisor ……………………..
Name of Supervisor: **Mr. HIMANSHU MITTAL**
Designation:
Date: 13/05/2017

(IV)

ACKNOWLEDGEMENT

We want to express our appreciation to our mentor **Mr. HIMANSHU MITTAL,** faculty at Jaypee Institute of Information Technology, India who gave us the golden opportunity to do this wonderful project on "Test Case Optimization Using Nature Inspired Algorithm" , and my earnest appreciation to him , for his empowering direction, constant consolation and supervision over the span of present work and furthermore helped us in doing a great deal of re-pursuit. We likewise wish to extend our gratitude to our batch mates for their canny remarks and helpful proposals to enhance the nature of this project work.

Signature: …………………….
Name: VIBHOR GUPTA
Enrollment Number: 9913103515
Date: 13/05/2017

Signature: …………………….
Name: NAMAN SHARMA
Enrollment Number: 9913103546
Date: 13/05/2017

Signature: …………………….
Name: AVNEET SINGH
Enrollment Number: 9913103676
Date: 13/05/2017

(V)

SUMMARY

Software Testing is used broadly in industries now a day for quality assurance. With the complex varieties of software developing continuously, checking that it behaves according to the expected levels of quality and reliability is becoming more crucial, and more difficult and expensive. One of the essential task in software engineering process is downsizing the effort of software testing and tries to make it to reduce the cost & time of development. I n this project we are trying to optimize the test cases by using natured inspired algorithms.

Bacteriologic algorithm is one of the algorithms that we have implemented in our project. Regression testing is the trying of programming so as to ensure that the change made on the program lines does not influences alternate parts of the product. Prioritization techniques are used to avoid the drawbacks that can occur when test case minimization is used to discard test cases.

Another algorithm that is implanted is the Differential evolution algorithm which is also a nature inspired algorithm. Differential Evolution (DE) is a strategy that advances an issue by iteratively attempting to enhance an applicant arrangement concerning a given measure of value. In our project we have compared the two algorithms on the basis of their working and the fault detection and the different ways in which optimize the test cases leaving behind the test cases that are best for us.

(VI)

LIST OF FIGURES

LIST OF TABLES

(VIII)

LIST OF SYMBOLS AND ACRONYMS

SYMBOLS

1. **% - Percentage**
2. **& - And**
3. **+ - Addition**
4. *** - Multiplication**
5. **< - Smaller than**
6. **- - Subtraction**
7. ∑ **- Summation**

ACRONYMS

1. **BA - Bacteriologic Algorithm**
2. **GA – Genetic Algorithm**
3. **DE – Differential Evolution Algorithm**
4. **SDLC – Software Development Life Cycle**
5. **FIT – Fitness Function**
6. **BC – Branch Coverage**
7. **FDA - Fault Detection**

Chapter 1 - Introduction

1.1 General Introduction

This paper manages programming experiment advancement utilizing Nature Inspired Algorithm and necessity mapping-based approach. Experiment improvement manages choosing viable experiments having most extreme code scope and blame discovery ability, consequently minimizing and organizing the experiments.

Regression test suites being too expansive to re-execute in given time, some approach ought to be connected to limit the quantity of experiments, at long last to decrease the cost and timing of testing procedure. This can be accomplished by utilizing at least one of the two strategies viz. experiment minimization or prioritization. Experiment minimization is decreasing the quantity of test suites by dispensing with repetitive experiments. Unnecessary test cases are removed from the test suite. A test case is redundant if same requirements can be fulfilled by other test cases. A test suite consists of all the test cases that satisfy all the testing requirements. Test case prioritization is reordering of the test suites according to an appropriate criterion like code coverage, branch detection, fault detection, etc.

1.2 Problem statement

To Research about test case generation, optimization and prioritization using Nature inspired algorithms like Bacteriologic Algorithm (BA), Differential Algorithm (DE) and hence create a tool for comparison between the above mentioned algorithm with respect to test case prioritization and optimization.

1.3 Details of Empirical Study

Programming advancement associations spend impressive part of their financial plan and time in testing related exercises. To carry out any occupation productively you should take in the specialty of experiment prioritization. This subjective and troublesome piece of testing is about test arranging, cost, esteem, and being logical about which tests to keep running with regards to your particular venture. Testing is a standout amongst the most basically critical periods of the product advancement life cycle and devours huge assets as far as exertion, time and cost.

Differential evolution (DE) is a method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. To solve a problem, this algorithm deals with different possible combination and tries to check the combination which fits the best or the solution that gives the best possible outcome.

Differential evolution is actually a specific subset of the broader space of genetic algorithms, with few restrictions like the genotype is some form of real-valued vector and the mutation /crossover operations make use of the difference between two or more vectors in the population to create a new vector (typically by adding some random proportion of the difference to one of the existing vectors, plus a small amount of random noise).

DE performs well for certain situations because the vectors can be considered to form a "cloud" that explores the high value areas of the solution space quite effectively. It's pretty closely related to particle swarm optimization in some senses. The capacity takes a hopeful arrangement as contention as a vector of genuine numbers and creates a genuine number as yield which shows the wellness of the given applicant arrangement.
The differential weight, crossover probabilities and the population size plays a major role in finding the best possible candidate solution. Variants of the DE algorithm are continually being developed in an effort to improve optimization performance.

1.4 Overview of proposed solution approach

The aim of this project is to lower the time and cost-spend on the Testing phase in Software Development Life Cycle (SDLC).
In this project we will generate the test cases by taking the range of the input variables and further combining them in order to create an initial population of test cases. The test cases will be further selected one by one from the initial population and will execute the program in order to check the coverage and branches by creating a branching matrix and fault coverage matrix as per Bacteriologic Algorithm.
Bacteriologic Algorithm is a special kind of Genetic Algorithm which uses its fit function and weights in order to prioritize the Test Cases. By using the fault coverage and branch matrix we will combine the test cases with each other in such a way that if total N test cases are there in the initial population then the total combinations of test cases from initial population will be N!.
The test cases combination hence created will be further scrutinized by calculating the fit function with the weights given by the user as input in order to prioritize the test cases. Only those test cases combination will be shown to the user that have 100% fit function and hence by 100% fit function value we mean that these test case combination are covering every part of the problem.
The benefit of this project will be that it will make testing part less costly and more over less efforts in terms of labor and time will be spent on it as it will make the testing semi-automatic plus will show only those test cases to the user that cover every part and fault of the problem given to it.
Differential evolution is actually a specific subset of the broader space of genetic algorithms, with few restrictions like the genotype is some form of real-valued vector and the mutation / crossover operations make use of the difference between two or more vectors in the population to create a new vector (typically by adding some random proportion of the difference to one of the existing vectors, plus a small amount of random noise).
The capacity takes a hopeful arrangement as contention as a vector of genuine numbers and creates a genuine number as yield which shows the wellness of the given applicant arrangement.
The differential weight, crossover probabilities and the population size plays a major role in finding the best possible candidate solution. Variations of the DE calculation are consistently being produced with an end goal to enhance streamlining execution. Various plans for per-

forming hybrid and transformation of operators are conceivable in the fundamental calculation.

1.5 Support for Novelty

Testing a Project is a crucial Phase and an area of constant efforts and cost. Testing takes around 47% and more of the budget of the whole project. Moreover 35% of the whole project time is spent in testing. We cannot eliminate this phase because of this importance but we can reduce its part as testing grabs a lot of portion from the SDLC. Therefore testing phase and techniques are under constant development and innovations.

1.6 Tabular Comparison

	DIFFERENTIAL EVOLUTION ALGORITHM	**BACTERIOLOGIC ALGORITHM**
1. Begin	Generate randomly an initial population of the solution.	Create arbitrarily an underlying population of the arrangement.
	Ascertain the wellness estimation of the underlying populace.	Figure the fitness value of the underlying populace.
2. Repetition	For each parent, select three solutions at random.	Select a pair of parents on the basis of the fitness value.
	Create one off-spring using DE operators.	Create two off-springs using crossover.
	Performs above step for the number of times equal to the population size.	Apply mutation to each child.
	For each member of the next generation (i) If off-spring(x) is more fit than the parent(x) (ii) Parent(x) is replaced.	Evaluate the mutated off-spring
	Continue to work until stop is condition is satisfied.	All the off-springs will be the new population, the parents will die.
		Continue to work until stop is condition is satisfied.

Table 1 (Comparison of the Algorithm's)

Chapter 2 - Literature Survey

2.1 Summary of relevant papers

1. **Title of paper:** Test Case Minimization Techniques: A Review
   Authors: Rajvir Singh and Mamta Santosh
   Year of Publication: 12, December – 2013
   Publishing details where this paper was published: International Journal of Engineering Research & Technology (IJERT)
   Summary: Test case minimization techniques are used to minimize the testing cost in terms of execution time, resources etc. The purpose of test case minimization is to generate representative set from test suite that satisfy all the requirements as original test suite with minimum number of test cease. Main purpose of test case minimization techniques is to remove test cases that become redundant and obsolete over time.
   This paper outlined the brief summary of techniques that has been proposed in literature for test case minimization. The techniques studied include Heuristic H, GRE, and Divide and conquer approach, Genetic algorithm, selective redundancy, Test Filter, Integer Linear Programming based DILP etc. Almost among these produced reduced test suites. Each technique is superior to another in some aspect. Many of them generated significant reduction in test suite, but it is harder to tell which one performs best. Heuristic based approach produced significant reduction but less fault detection effectiveness. ILP based approach guaranteed minimal set but more complex and increased cost. For a technique to be efficient it should be good in both - reduced test suite size and improved fault detection efficiency.
   Weblink:
   https://www.researchgate.net/265086128_Test_Case_Minimization_Techniques_A_R.

2. **Title of paper:** Test case optimization a nature inspired approach using bacteriologic algorithm.
   Authors: Praveen Ranjan Srivastava
   Year of Publication: 2016
   Publishing details where this paper was published: International Journal of Bio-Inspired computation
   Summary: This paper recommends an approach for the experiment minimization and prioritization utilizing a BA and prerequisite mapping system for enhancing the blame location capacity of the experiments. Experiment minimization manages diminishing the quantity of test suites by disposing of excess experiments. Experiment prioritization is reordering of the test suites as indicated by a fitting basis like code scope, branch discovery, blame identification, and so on.
   Necessity mapping is utilized to delineate experiments with different prerequisites removed from code like branch scope, variable def-utilize, and so on. The frameworks gotten from this procedure will be utilized to decide beginning test suite's size and wellness figuring in BA. The wellness work, fit utilized here for ascertaining wellness of test suites is the met-

ric induced as whole of result of separate weights with bc and fda which can be straightforwardly gotten from the two tables shaped by necessity mapping.
fit = w1 * bc + w2 * fda
where bc is branch scope (gotten from branch scope table)
also, fda is blame discovery ability(obtained from blame scope table).
w1 and w2 are weights who's whole will extend from 0 to 1. They delineate the criticalness of every metric which will change for individual program.
Web link: http://www.inderscienceonline.com/doi/abs/10.1504/IJBIC.2016.076335

3. **Title of paper:** A Review on Automated Test Case Generation Using Genetic Algorithms
Authors: Rijwan Khan and Dr. Mohd Amjad
Year of Publication: 12, December – 2014
Publishing details where this paper was published: International Journal of Advanced Research in Computer Science and Software Engineering
Summary: Testing is relying upon the experiments (contribution for the product). So to locate the appropriate experiments for programming is a troublesome errand. Arbitrarily created test case sets aside a great deal of opportunity to test the product. To diminish the season of the testing procedure programmed experiment era is a decent alternative. In programmed experiment era, hereditary calculation assumes a vital part. With the utilization of the hereditary calculation we can create reasonable experiments.
The key issue in programming testing is to produce experiment and its mechanization. It enhances the proficiency and viability and brings down the high cost of programming testing .Simple random technique is insufficient to create satisfactory measure of test information. Along these lines, there is a requirement for producing test information utilizing look based systems.
Various enhancement systems have been connected for experiment era however nobody could accomplish the best execution for each bit of code. Since, experiment era has turned into an improvement issue thus scope stays open to apply some more strategies to accomplish better outcomes. In this paper, through talk about GA as enhancement systems for experiment era is secured, which will clear the way for further work toward this path.
Weblink: https://www.ijarcsse.com/docs/papers/Volume_4/12.../V4I12-0265.pdf

4. **Title of paper:** An evaluation of Differential Evolution in Software Test Data Generation
Authors: R. Landa Becerra, R. Sagarna and X. Yao
Year of Publication: 24, October – 2009
Publishing details where this paper was published: International Journal of Advanced Research in Computer Science and Software Engineering
Summary: In this paper, they have connected DE to the issue of creating an arrangement of test data sources that cover all the branches in the source code of a program. Expanding upon a novel requirement taking care of detailing, we have assessed exactly the execution of various famous DE models. Additionally, we have confronted DE with a notable Genetic Algorithm, i.e. BGA, and other test information generators in the writing.
The outcomes acquired by DE are better in the vast majority of the issues than those gotten by BGA, yet legitimately setting the parameters is a vital assignment. DE shows up uniquely sensi-tive to the estimation of F. Parameters as G and M just should be sufficiently ex-

tensive to permit DE play out the inquiry as needs be with the trouble of the issue. Joining focal points of various variations, is a conceivable future work which may enhance the execution of DE in this field. The utilization of components which adaptively give this blend, by incorporating data separated from the issue, are uniquely appropriate. In this way, a social algorithm system is a decent possibility for future research.

Weblink:
https://pdfs.semanticscholar.org/0715/86d3fe8061843482d6679fe4647c8c3ab8af.pdf

5. **Title of paper**: Using Genetic Algorithms and Dominance Concepts for Generating Reduced Test Data

   Authors: Ahmed S. Ghiduk, Moheb R. Girgis

   Year of Publication: March 6, 2008

   Publishing details where this paper was published: Informatica **34** (2010) 377–385

   Summary: This paper exhibited a programmed test-information era method that uses a hereditary calculation. Tests have been done to assess the adequacy of the proposed GA system contrasted with the RT strategy, and to assess the viability of the new wellness work and the procedure used to lessen the cost of programming testing. The consequences of these trials demonstrated that the proposed GA method beat the RT procedure in 7 out of the 9 programs utilized as a part of the tests. In the other two projects, the proposed GA achieved an indistinguishable scope rate from the RT system. The investigations additionally demonstrated that the proposed procedure lessened the cost of programming testing by over 75%. Likewise, the consequences of the investigations demonstrated that the new wellness capacity is very reasonable to assess the created test-information and demonstrated the convenience of the ideas of predominance relations between hubs of the program's control stream diagram in reducing the quantity of test prerequisites. This system is being changed to produce test information for information stream testing. The ideas of strength relations between hubs of the professional gram's control stream diagram will be utilized to characterize another wellness capacity to assess the produced test information for information stream testing.

   Weblink: www.informatica.si/index.php/informatica/article/view/312

6. **Title of paper:** A Comparative Study on Differential Evolution and Genetic Algorithms for some Combinatorial Problems

   Authors: Brian Hegerty, Chih-Cheng Hung, and Kristen Kasprak

   Year of Publication: 24, October – 2009

   Publishing details where this paper was published: Southern Polytechnic State Univer sity, Marietta GA 30060,USA,

   Summary: This paper looks at the execution of streamlining systems, differential advancement and hereditary calculation, for taking care of some combinatorial issues. The voyaging salesperson issue and the N-Queens issue are both exemplary cases of a combinatorial issue which is NP-Complete. In both issue spaces the Genetic Algorithm's outcomes were less significant than those of Differential Evolution's outcomes. Be that as it may, there is an incentive in the relative speed of the Genetic Algorithm's outcomes in the Traveling Salesman Problem if a nearby least will be adequate. In the Traveling Salesman

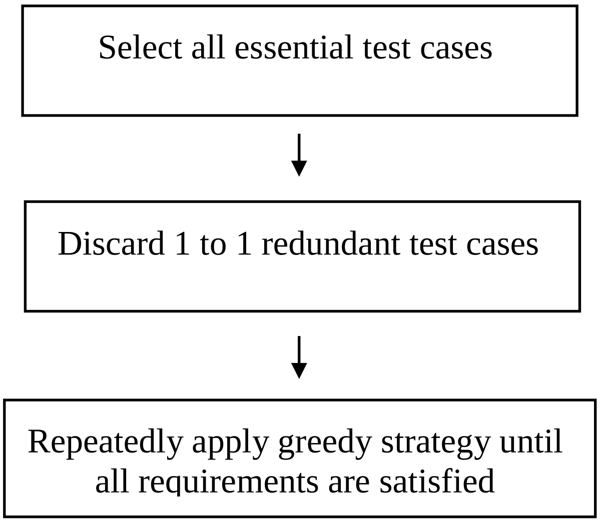
Problem both calculations created great outcomes rapidly, however the Differential Evolutionary Algorithm kept on enhancing the voyage through the urban communities. It is vital to note that with bigger N values the number of eras was bigger for hereditary calculation however the running time was practically identical and now and again superior to differential advancement. This is because of the overhead brought about by the calculation unpredictability of differential development, in reality an estimation of an opportunity to finish a solitary era for Differential Evolution is all things considered four times bigger than it takes to finish a comparable era (in light of populace size and issue measure) for the Genetic Algorithm. In any case, for bigger populace sizes it appears that differential development can beat the hereditary calculation in both eras and runtime, as can be seen with the outcomes from N= 20, populace sizes = 100, 500, 1000. These actualities improve Differential Evolution approach an answer for these combinatorial issues.
Weblink: https://pdfs.semanticscholar.org/cc44/e4e0dd2de316a9a19f1480b2772b969d4715.pdf

2.2 Integrated summary

BACTERIOLOGIC ALGORITHM

In Bacteriologic Algorithm we had taken some random inputs from the user and we have calculated the branch coverage and the fault detection value of each test case. Then the fitness value is calculated for each test case. Some pairs of test cases are formed at random and new test cases are generated by crossover of original test cases and these test cases are the child and the original test cases are the parent. Then the fitness value of these test cases are calculated and if the value of the fitness value of the children is greater than that of the parent then the parent test cases are replaced by children test cases. Also the test cases go through some other processes like minimization and prioritization which also helps in improving the test suites making them more efficient. Regression testing is performed on these test cases in which all of the possible crossover are done and then the test cases with the highest value of the fitness function are taken which improves the accuracy and the efficiency.

Figure 1 (Bacteriologic Algorithm)

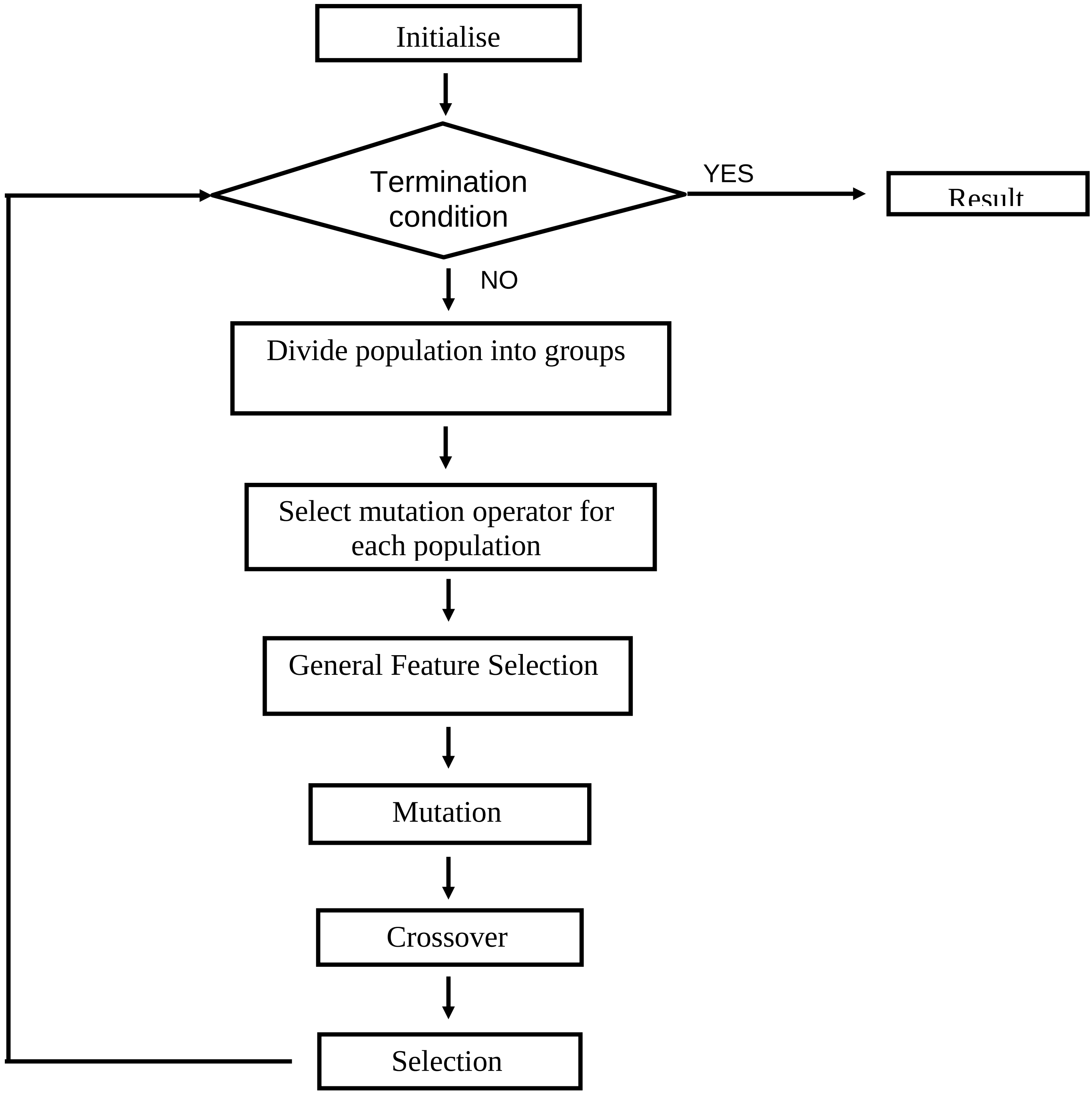


DIFFERENTAL EVOLUTION ALGORITHM

Differential Evolution is a method that optimizes problem iteratively and make an effort to make better candidate solution.DE being a meta-heuristic black box technique optimizes any problem by enhancing the initial population by replacing the initial population with better solutions and maintains it of the same size on the basis of a formula.
DE is an optimization technique which iteratively modifies a population of candidate solutions to make it converge to an optimum of your function. The DE algorithm gives us a description of or can be delineated as an evolutionary type, random optimization algorithm. The quality of DE algorithm over others is that it is easy structure not at all complicated and robust. To the same degree all evolutionary algorithms, it runs or functions by applying a set or population P = {P1, P2, ..., PN} of possible answer to search the solution space. The population of size (N), remains continuous everywhere. In every extension the algorithm goal is to

make fresh or newborn population by putting in place of points in the present population *P* with improved points. The choicest or most essential part or most vital part of some idea that the population is merely a set of points $P_{k,j}$ where *k* represents the indices of the member's in the initially generated population of test cases and *j* representing the iteration of the population to it belongs. Each $P_{k,j}$ consists of *n* components, where *n* being the dimension of the represented problem. By a recurrent procedure of reproduction (mutation and crossover) and selection, the population N is used to determine the direction of the global minimum.

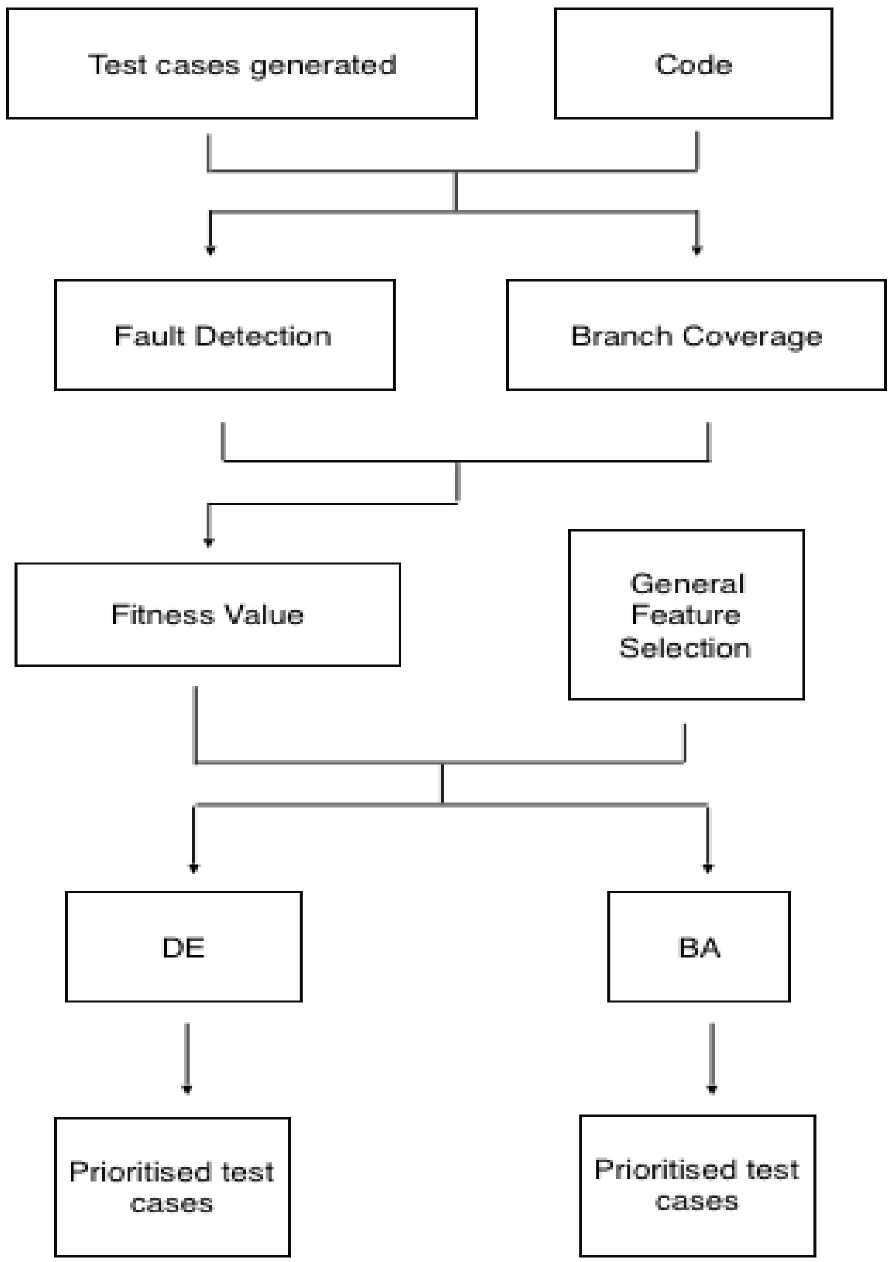
Figure 2 Differential Evolution Flow Chart



Chapter 3 - Analysis, Design and Modeling

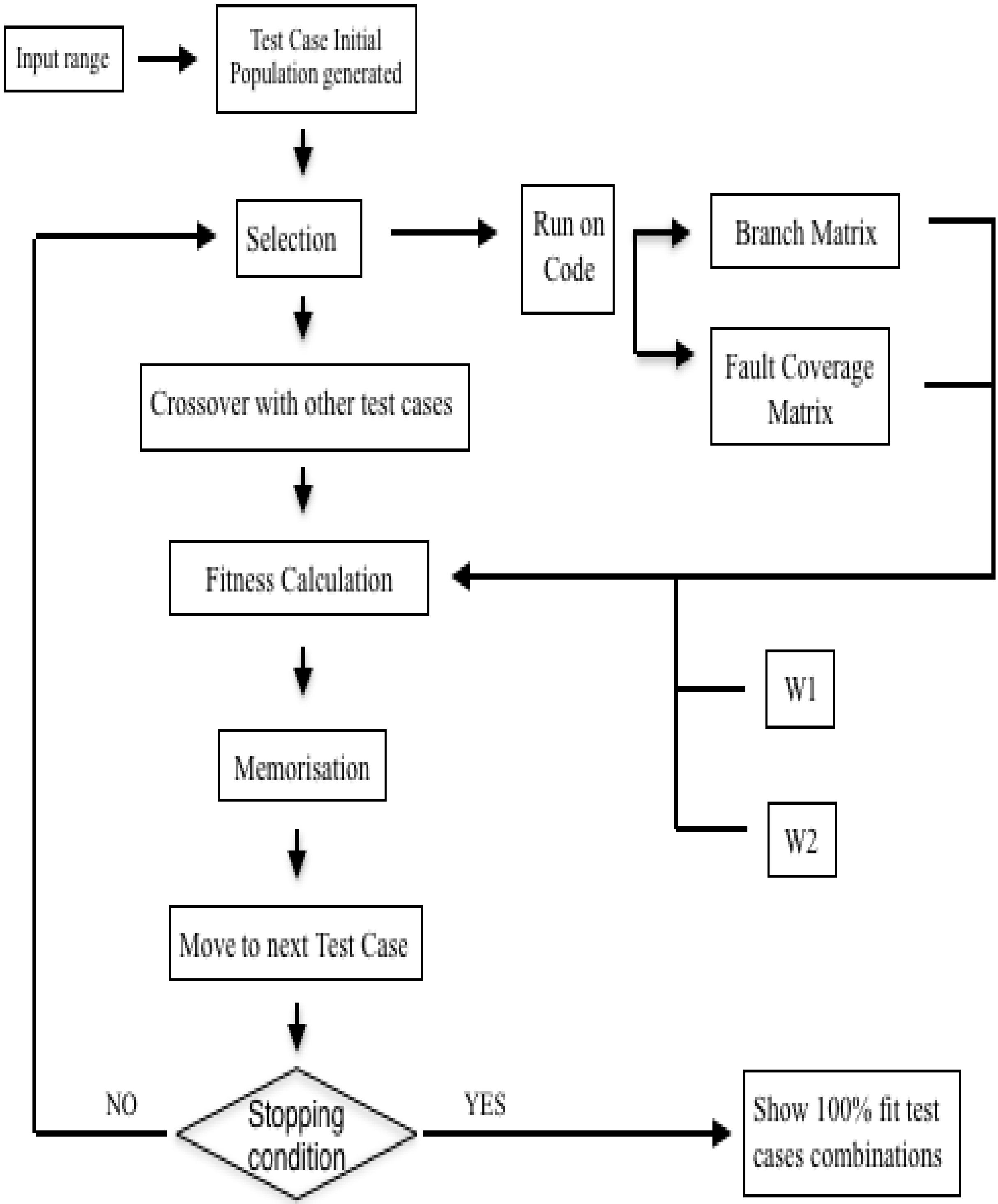
3.1 Overall description of the project

Figure 3: Basic overview



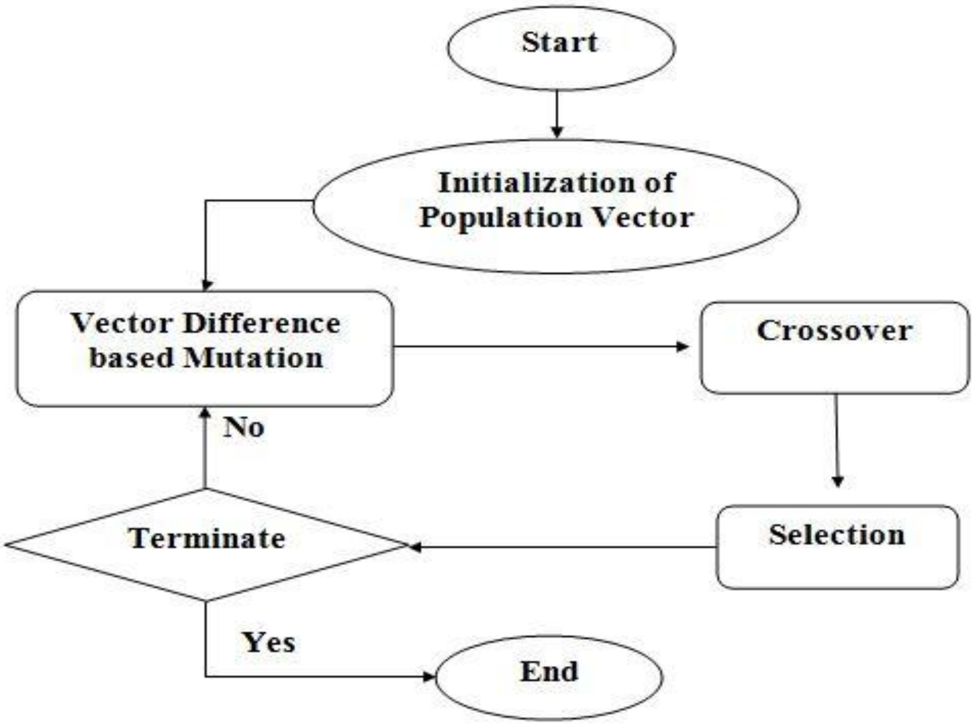
BACTERIOLOGIC ALGORITHM

Figure 4 (Code Layout Of Bacteriologic Algorithm)



DIFFERENTIAL EVOLUTION ALGORITHM

Figure 5 (Code Layout Of Differential Algorithm)



3.2 Functional Requirements

1. Input specification must specify the inputs that are required to execute the test case.
2. Output specification must specify expected output from the test case execution to decide pass or failure.
3. The Product should be able to handle complex and large quantity of data.
4. The Proposed algorithm should perform better than the present algorithm.
5. The Product should be able to optimize the test cases and prioritize them according to different problems given to it.

3.3 Non-Functional Requirements:-

1. The Data shown to the user should be easily understandable by him.
2. Reliability on the product and improved algorithm should be guaranteed.
3. Performance of the improved algorithm and the tool must be satisfactory.
4. Data integrity should be there for the critical variables.
5. The product is usable within almost any environment. That means most systems should be able to quickly run and load the product with little difficulty.

3.4 Logical Database Requirements

Minimum Software Requirement

The product is developed on Netbeans and Wamp server and requires the knowledge of JAVA and DBMS. The requirements for these softwares are as follows:-

- NetBeans bundles only require the Java Runtime Environment (JRE) 7 or JRE 8 to be installed and run.
- Java features in the IDE require JDK 7 or JDK 8. JavaFX 2.2 (or newer) features require JDK 7 Update 6 (or newer).

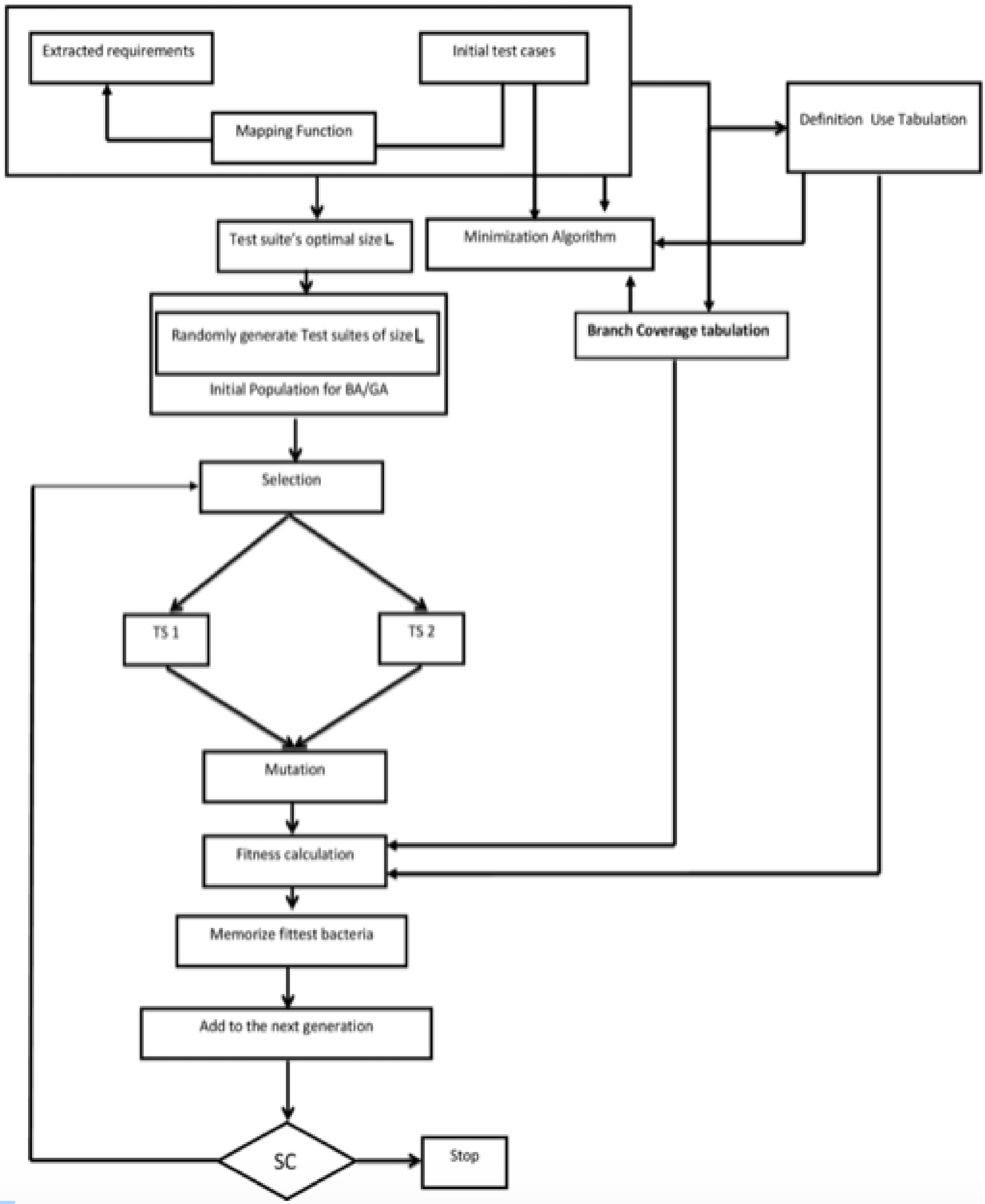
Minimum Hardware Requirement

- Microsoft Windows XP Professional SP3/Vista SP1/Windows 7 Professional:
    - Processor: 800MHz Intel Pentium III or equivalent
    - Memory: 512 MB
    - Disk space: 750 MB of free disk space
- Ubuntu 9.10:
    - Processor: 800MHz Intel Pentium III or equivalent
    - Memory: 512 MB
    - Disk space: 650 MB of free disk space
- Macintosh OS X 10.7 Intel:
    - Processor: Dual-Core Intel
    - Memory: 2 GB
    - Disk space: 650 MB of free disk space

3.5 Design Documentation

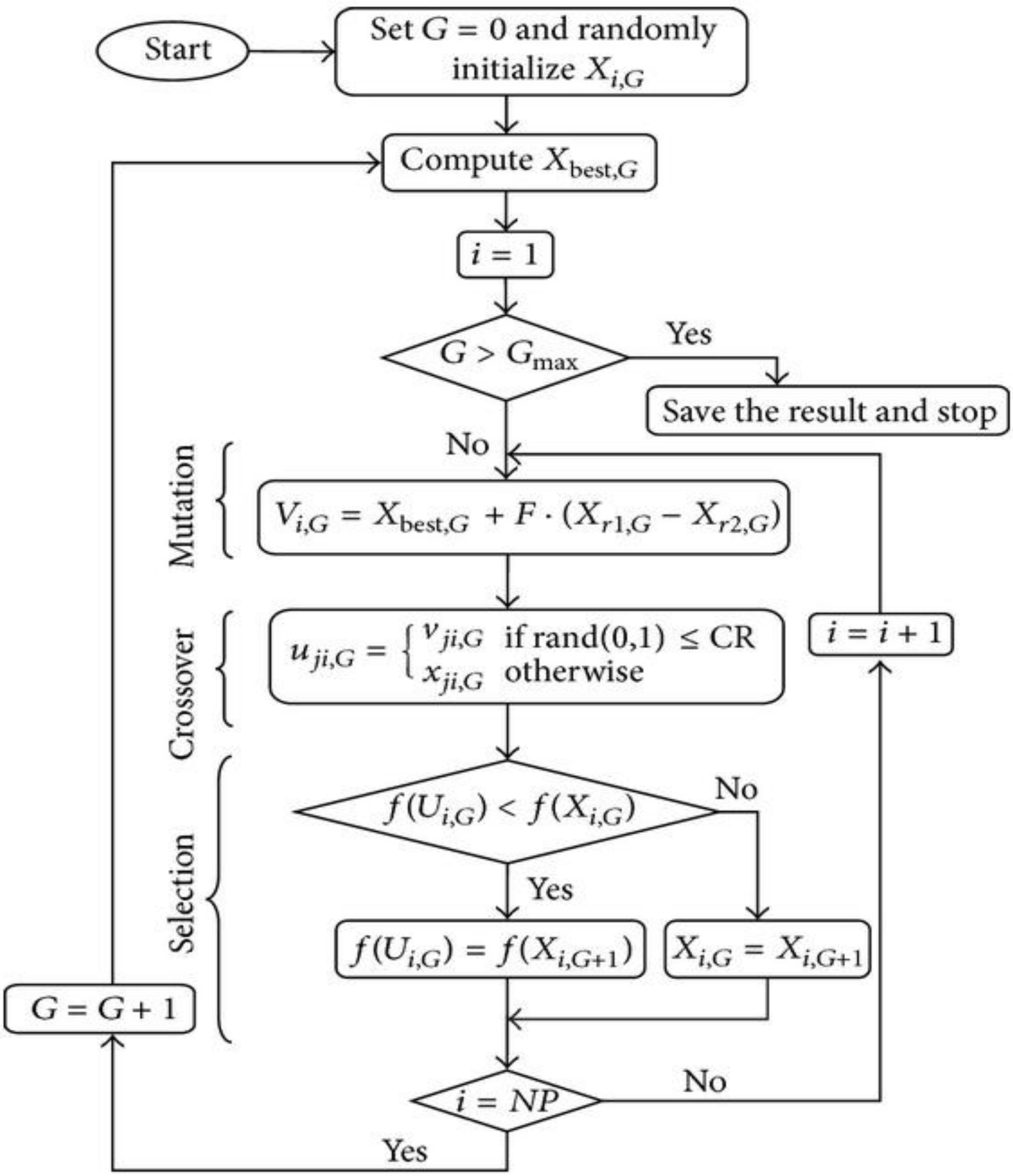
BACTERIOLOGIC ALGORITHM

Figure 6 (Bacteriologic Algorithm Design)



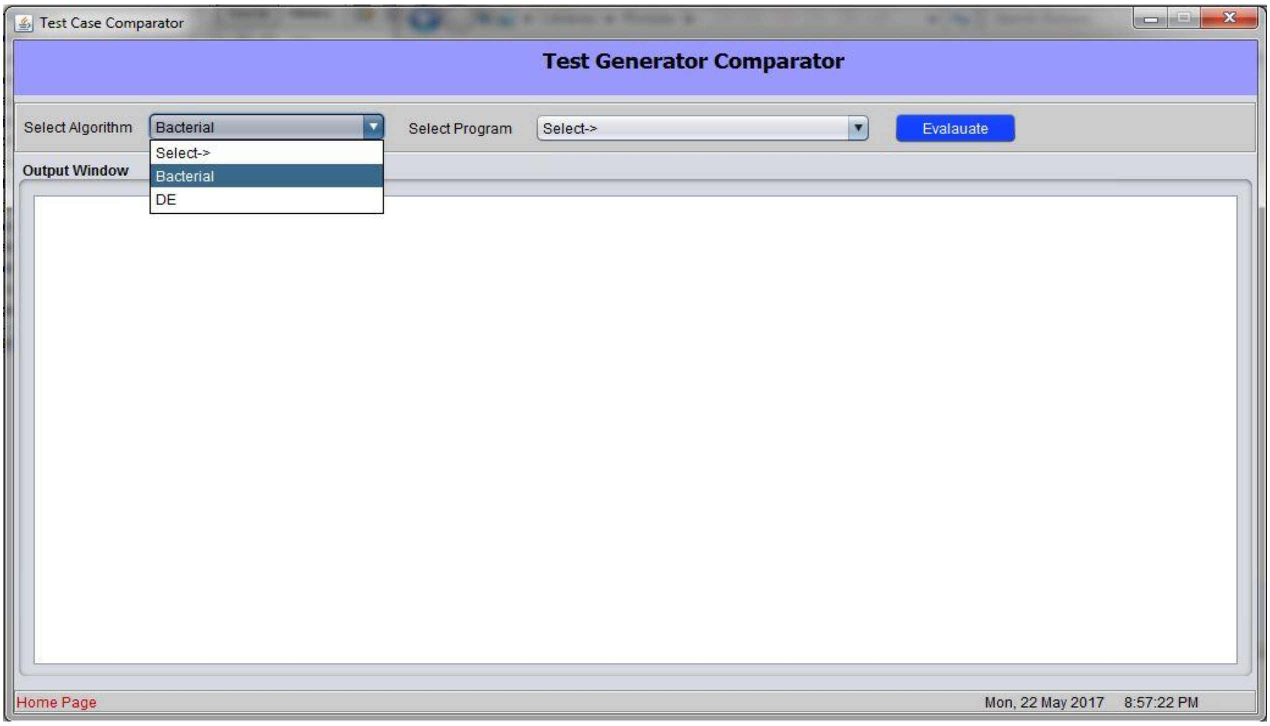
DIFFERENTIAL EVOLUTION ALGORITHM

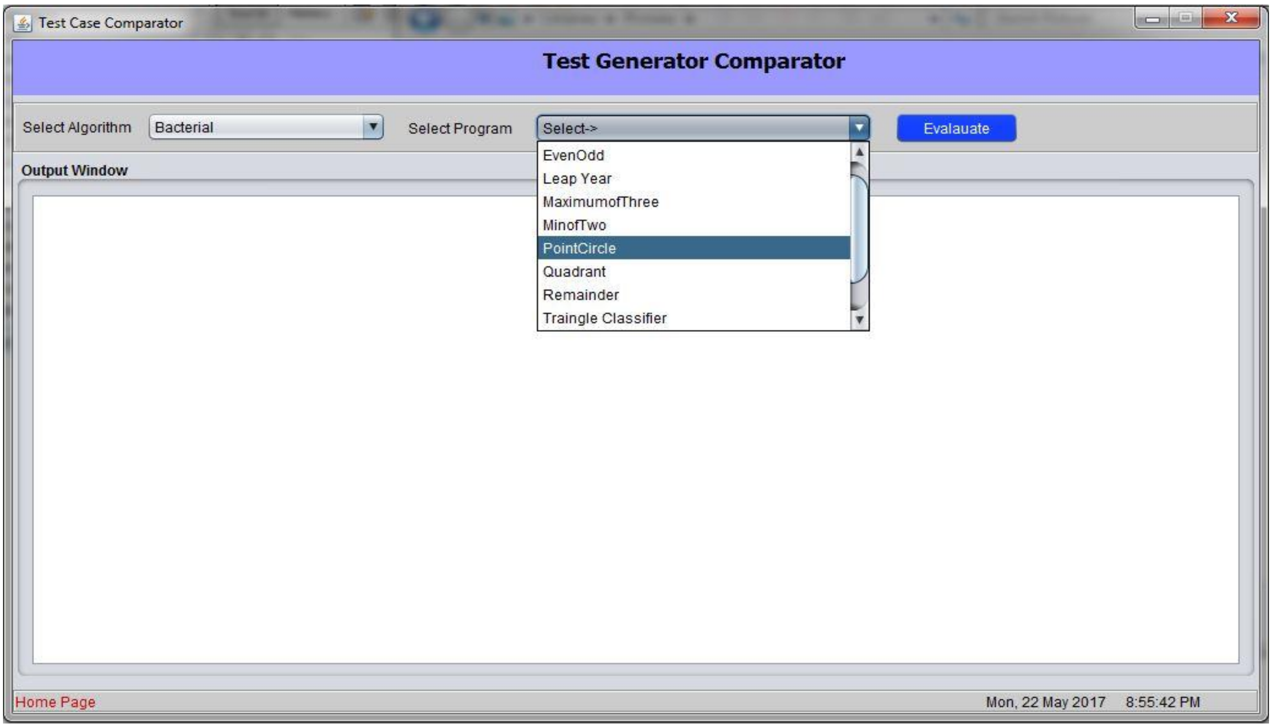
Figure 7 (Differential Evolution Algorithm Design) [14]

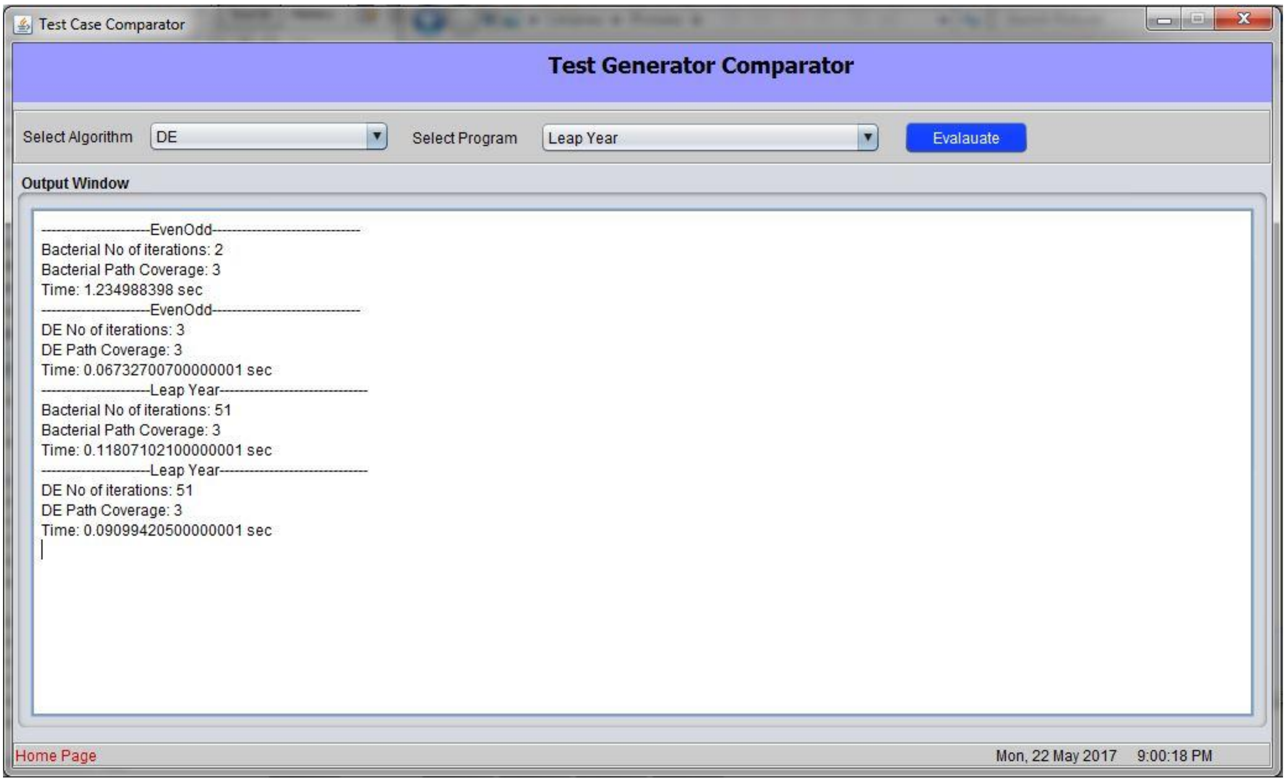


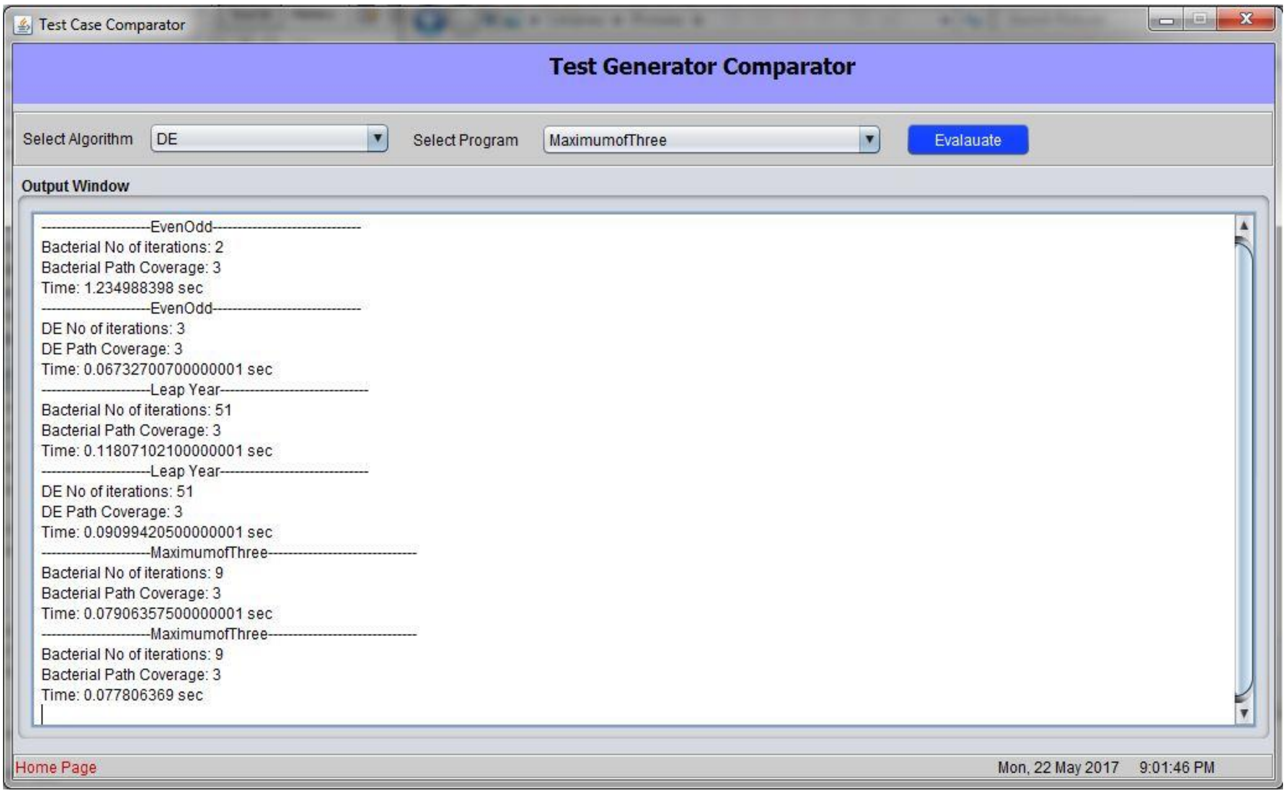
Chapter 4 - Implementation details and issues

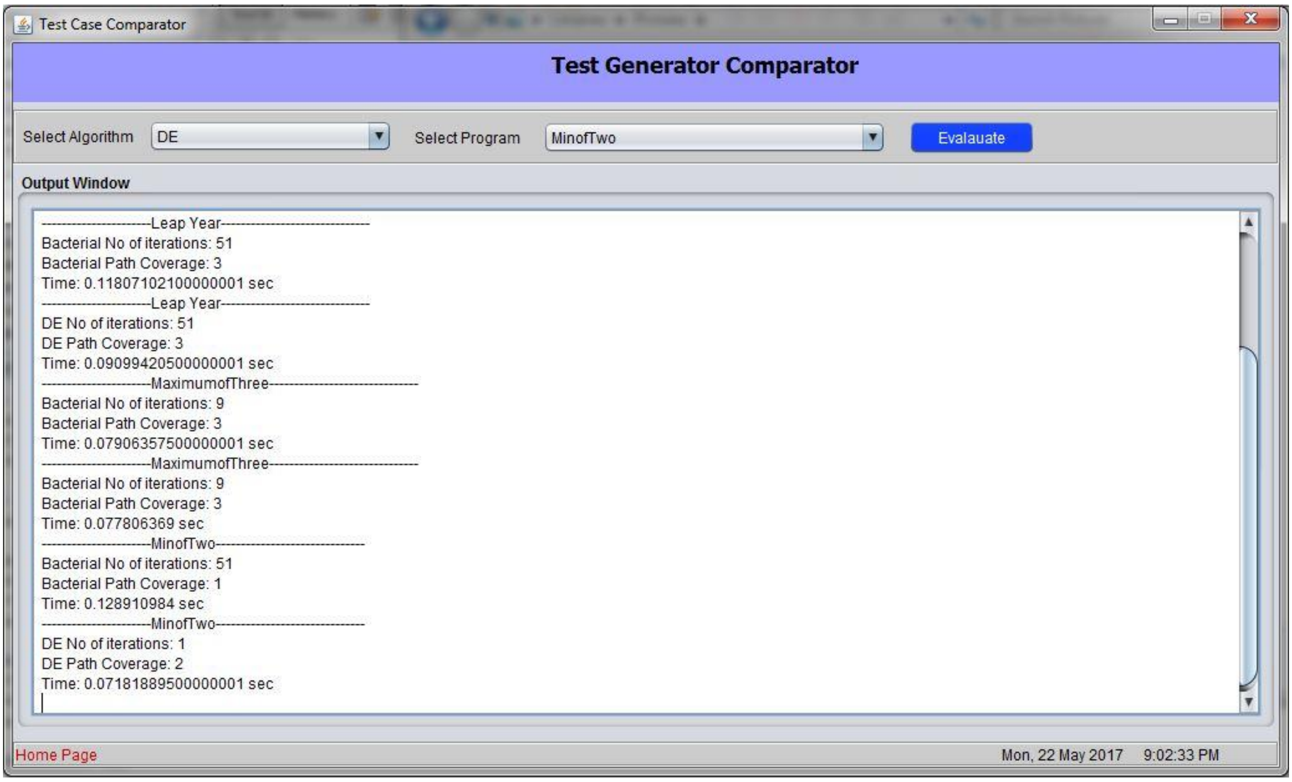
4.1 Implementation details

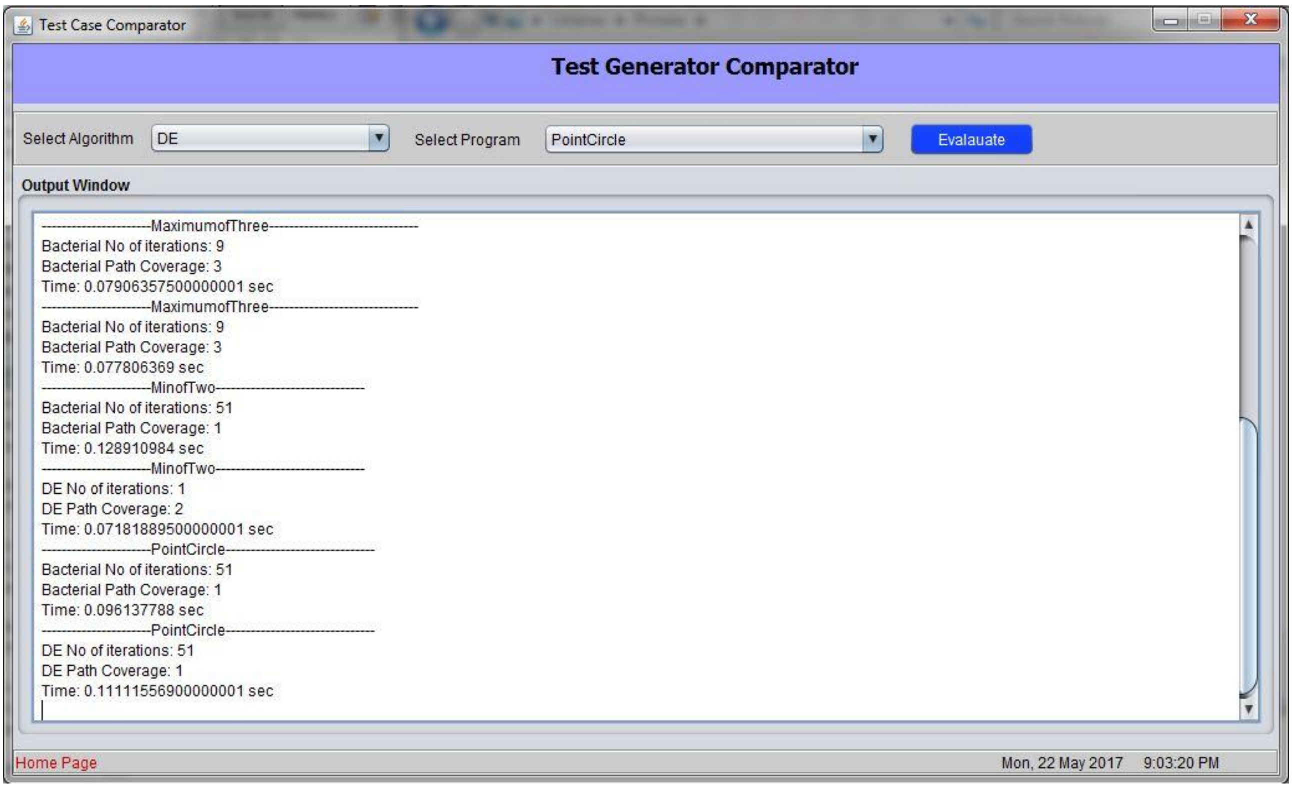


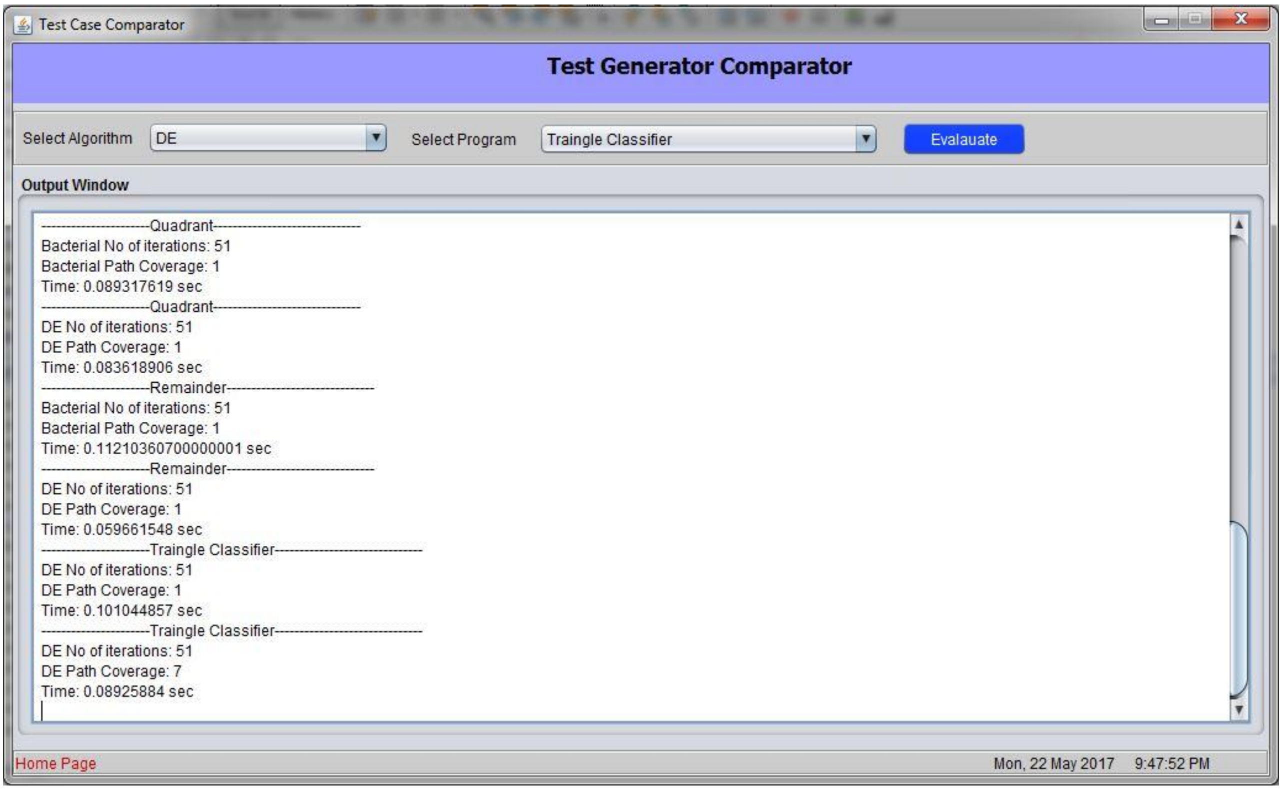


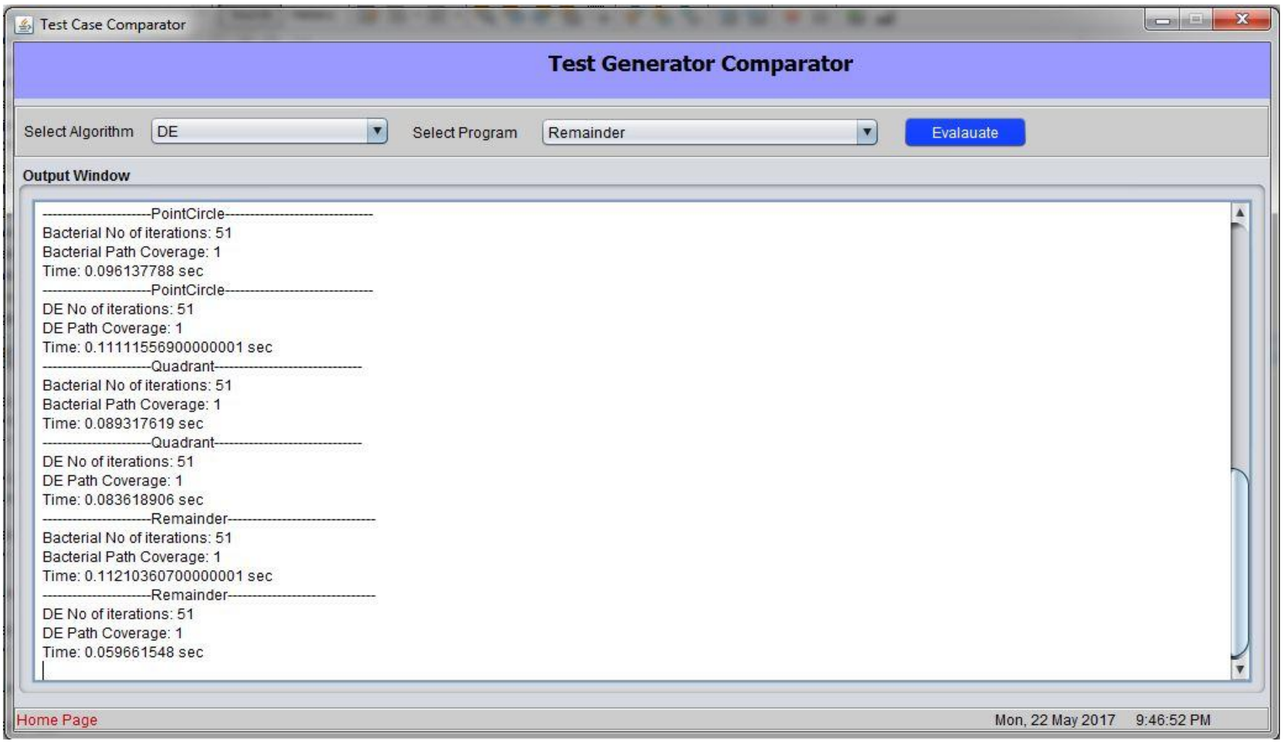

Test Case Comparator
Test Generator Comparator
Select Algorithm
DE
Select Program
Leap Year
Evalauate
Output Window
---------------------EvenOdd---------------------------
Bacterial No of iterations: 2
Bacterial Path Coverage: 3
Time: 1.234988398 sec
---------------------EvenOdd---------------------------
DE No of iterations: 3
DE Path Coverage: 3
Time: 0.06732700700000001 sec
---------------------Leap Year---------------------------
Bacterial No of iterations: 51
Bacterial Path Coverage: 3
Time: 0.11807102100000001 sec
---------------------Leap Year---------------------------
DE No of iterations: 51
DE Path Coverage: 3
Time: 0.09099420500000001 sec
Home Page
Mon, 22 May 2017 9:00:18 PM

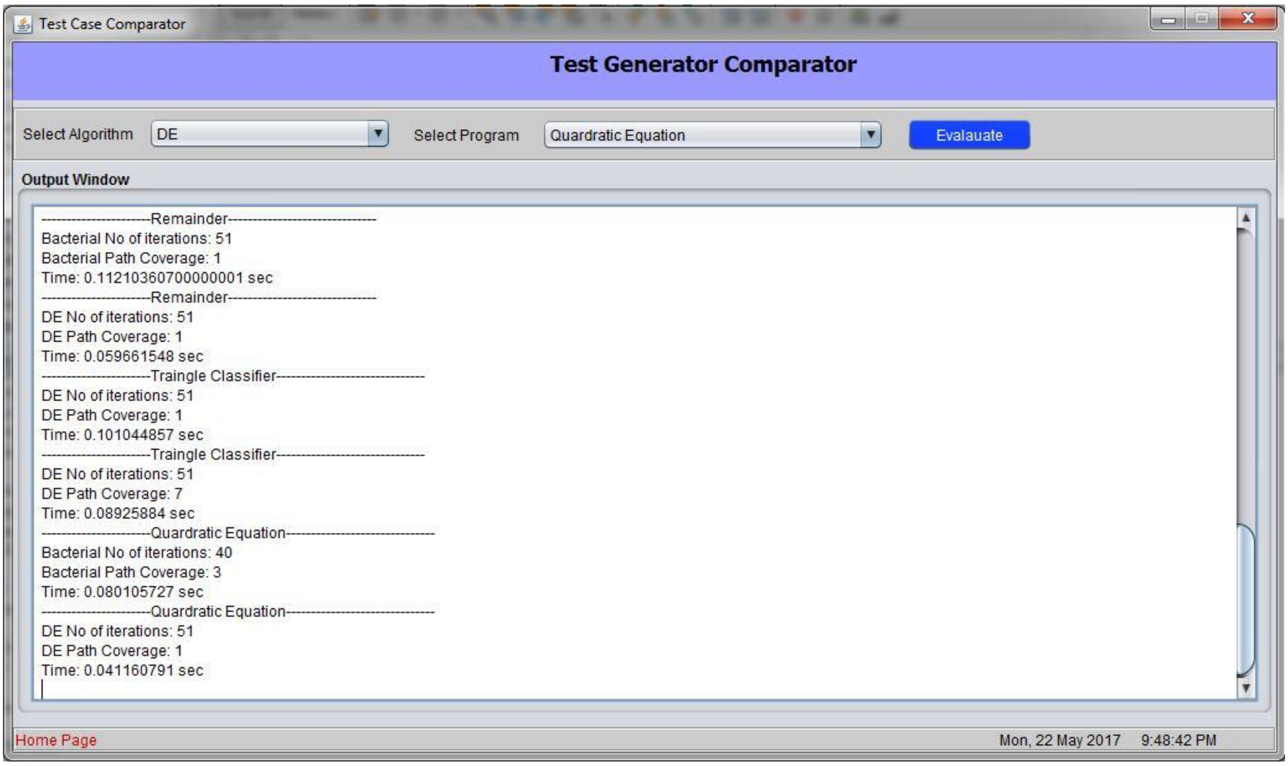

Test Case Comparator
Test Generator Comparator
Select Algorithm
DE
Select Program
MaximumofThree
Evalauate
Output Window
---------------------EvenOdd---------------------------
Bacterial No of iterations: 2
Bacterial Path Coverage: 3
Time: 1.234988398 sec
---------------------EvenOdd---------------------------
DE No of iterations: 3
DE Path Coverage: 3
Time: 0.06732700700000001 sec
---------------------Leap Year---------------------------
Bacterial No of iterations: 51
Bacterial Path Coverage: 3
Time: 0.11807102100000001 sec
---------------------Leap Year---------------------------
DE No of iterations: 51
DE Path Coverage: 3
Time: 0.09099420500000001 sec
---------------------MaximumofThree---------------------------
Bacterial No of iterations: 9
Bacterial Path Coverage: 3
Time: 0.07906357500000001 sec
---------------------MaximumofThree---------------------------
Bacterial No of iterations: 9
Bacterial Path Coverage: 3
Time: 0.077806369 sec
Home Page
Mon, 22 May 2017 9:01:46 PM

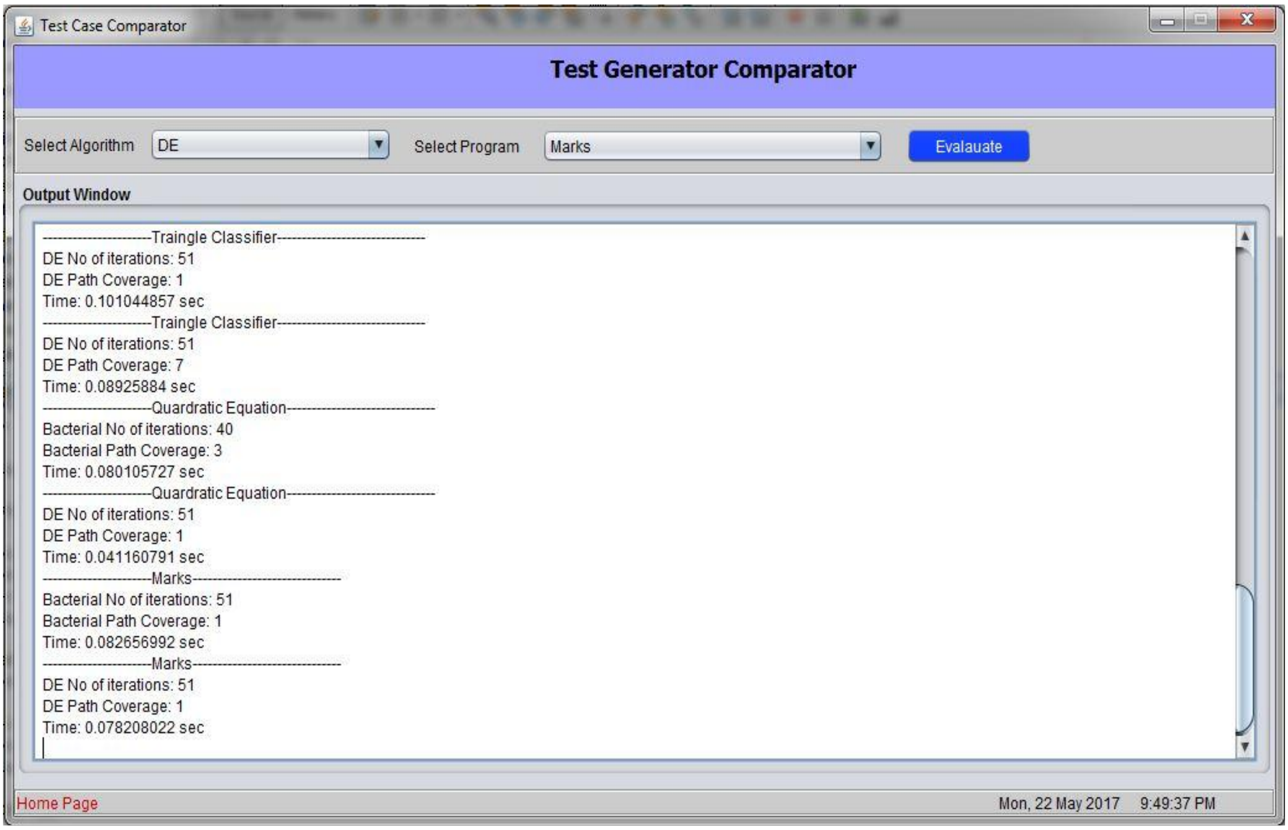

Test Case Comparator
Test Generator Comparator
Select Algorithm
DE
Select Program
MinofTwo
Evalauate
Output Window
--------------------Leap Year---------------------------
Bacterial No of iterations: 51
Bacterial Path Coverage: 3
Time: 0.11807102100000001 sec
--------------------Leap Year---------------------------
DE No of iterations: 51
DE Path Coverage: 3
Time: 0.09099420500000001 sec
--------------------MaximumofThree---------------------------
Bacterial No of iterations: 9
Bacterial Path Coverage: 3
Time: 0.07906357500000001 sec
--------------------MaximumofThree---------------------------
Bacterial No of iterations: 9
Bacterial Path Coverage: 3
Time: 0.077806369 sec
--------------------MinofTwo---------------------------
Bacterial No of iterations: 51
Bacterial Path Coverage: 1
Time: 0.128910984 sec
--------------------MinofTwo---------------------------
DE No of iterations: 1
DE Path Coverage: 2
Time: 0.07181889500000001 sec
Home Page
Mon, 22 May 2017 9:02:33 PM


Test Case Comparator
Test Generator Comparator
Select Algorithm
DE
Select Program
PointCircle
Evalauate
Output Window
--------------------MaximumofThree---------------------------
Bacterial No of iterations: 9
Bacterial Path Coverage: 3
Time: 0.07906357500000001 sec
--------------------MaximumofThree---------------------------
Bacterial No of iterations: 9
Bacterial Path Coverage: 3
Time: 0.077806369 sec
--------------------MinofTwo---------------------------
Bacterial No of iterations: 51
Bacterial Path Coverage: 1
Time: 0.128910984 sec
--------------------MinofTwo---------------------------
DE No of iterations: 1
DE Path Coverage: 2
Time: 0.07181889500000001 sec
--------------------PointCircle---------------------------
Bacterial No of iterations: 51
Bacterial Path Coverage: 1
Time: 0.096137788 sec
--------------------PointCircle---------------------------
DE No of iterations: 51
DE Path Coverage: 1
Time: 0.11111556900000001 sec
Home Page
Mon, 22 May 2017 9:03:20 PM


Test Case Comparator
Test Generator Comparator
Select Algorithm
DE
Select Program
Traingle Classifier
Evalauate
Output Window
---------------------Quadrant---------------------------
Bacterial No of iterations: 51
Bacterial Path Coverage: 1
Time: 0.089317619 sec
---------------------Quadrant---------------------------
DE No of iterations: 51
DE Path Coverage: 1
Time: 0.083618906 sec
---------------------Remainder---------------------------
Bacterial No of iterations: 51
Bacterial Path Coverage: 1
Time: 0.11210360700000001 sec
---------------------Remainder---------------------------
DE No of iterations: 51
DE Path Coverage: 1
Time: 0.059661548 sec
---------------------Traingle Classifier---------------------------
DE No of iterations: 51
DE Path Coverage: 1
Time: 0.101044857 sec
---------------------Traingle Classifier---------------------------
DE No of iterations: 51
DE Path Coverage: 7
Time: 0.08925884 sec
Home Page
Mon, 22 May 2017 9:47:52 PM


Test Case Comparator
Test Generator Comparator
Select Algorithm
DE
Select Program
Remainder
Evalauate
Output Window
---------------------PointCircle---------------------------
Bacterial No of iterations: 51
Bacterial Path Coverage: 1
Time: 0.096137788 sec
---------------------PointCircle---------------------------
DE No of iterations: 51
DE Path Coverage: 1
Time: 0.11111556900000001 sec
---------------------Quadrant---------------------------
Bacterial No of iterations: 51
Bacterial Path Coverage: 1
Time: 0.089317619 sec
---------------------Quadrant---------------------------
DE No of iterations: 51
DE Path Coverage: 1
Time: 0.083618906 sec
---------------------Remainder---------------------------
Bacterial No of iterations: 51
Bacterial Path Coverage: 1
Time: 0.11210360700000001 sec
---------------------Remainder---------------------------
DE No of iterations: 51
DE Path Coverage: 1
Time: 0.059661548 sec
Home Page
Mon, 22 May 2017 9:46:52 PM


Test Case Comparator
Test Generator Comparator
Select Algorithm
DE
Select Program
Quardratic Equation
Evalauate
Output Window
--------------------Remainder---------------------------
Bacterial No of iterations: 51
Bacterial Path Coverage: 1
Time: 0.11210360700000001 sec
--------------------Remainder---------------------------
DE No of iterations: 51
DE Path Coverage: 1
Time: 0.059661548 sec
--------------------Traingle Classifier---------------------------
DE No of iterations: 51
DE Path Coverage: 1
Time: 0.101044857 sec
--------------------Traingle Classifier---------------------------
DE No of iterations: 51
DE Path Coverage: 7
Time: 0.08925884 sec
--------------------Quardratic Equation---------------------------
Bacterial No of iterations: 40
Bacterial Path Coverage: 3
Time: 0.080105727 sec
--------------------Quardratic Equation---------------------------
DE No of iterations: 51
DE Path Coverage: 1
Time: 0.041160791 sec
Home Page
Mon, 22 May 2017
9:48:42 PM


Test Case Comparator
Test Generator Comparator
Select Algorithm
DE
Select Program
Marks
Evalauate
Output Window
--------------------Traingle Classifier---------------------------
DE No of iterations: 51
DE Path Coverage: 1
Time: 0.101044857 sec
--------------------Traingle Classifier---------------------------
DE No of iterations: 51
DE Path Coverage: 7
Time: 0.08925884 sec
--------------------Quardratic Equation---------------------------
Bacterial No of iterations: 40
Bacterial Path Coverage: 3
Time: 0.080105727 sec
--------------------Quardratic Equation---------------------------
DE No of iterations: 51
DE Path Coverage: 1
Time: 0.041160791 sec
--------------------Marks---------------------------
Bacterial No of iterations: 51
Bacterial Path Coverage: 1
Time: 0.082656992 sec
--------------------Marks---------------------------
DE No of iterations: 51
DE Path Coverage: 1
Time: 0.078208022 sec
Home Page
Mon, 22 May 2017
9:49:37 PM

id	algo	path	program
45	Bacterial	1->2->3->8->	EvenOdd
46	Bacterial	1->2->4->6->7->8->	EvenOdd
47	Bacterial	1->2->4->5->8->	EvenOdd
48	DE	1->2->3->8->	EvenOdd
49	DE	1->2->4->6->7->8->	EvenOdd
50	DE	1->2->4->5->8->	EvenOdd

id	algo	path	program
51	Bacterial	1->2->12->13->14->	Leap Year
52	Bacterial	1->2->3->9->10->11->14->	Leap Year
53	Bacterial	1->2->3->4->6->7->8->11->14->	Leap Year
54	Bacterial	1->2->3->4->5->8->11->14->	Leap Year
55	DE	1->2->12->13->14->	Leap Year
56	DE	1->2->3->9->10->11->14->	Leap Year
57	DE	1->2->3->4->6->7->8->11->14->	Leap Year
58	DE	1->2->3->4->5->8->11->14->	Leap Year

id	algo	path	program
59	Bacterial	1->2->3->8->	MaximumofThree
60	Bacterial	1->2->4->5->8->	MaximumofThree
61	Bacterial	1->2->4->6->7->8->	MaximumofThree
62	DE	1->2->4->6->7->8->	MaximumofThree
63	DE	1->2->3->8->	MaximumofThree
64	DE	1->2->4->5->8->	MaximumofThree

id	algo	path	program
65	Bacterial	1->2->3->4->5->6->9->	MinofTwo
66	Bacterial	1->2->3->4->5->7->8->9->	MinofTwo
67	DE	1->2->3->4->5->7->8->9->	MinofTwo
68	DE	1->2->3->4->5->6->9->	MinofTwo

id	algo	path	program
85	Bacterial	1->2->4->5->6->	Remainder
86	Bacterial	1->2->3->6->	Remainder

id	algo	path	program
69	Bacterial	1->2->4->6->8->9->10->	PointCircle
70	Bacterial	1->2->3->10->	PointCircle
71	Bacterial	1->2->4->5->10->	PointCircle
72	Bacterial	1->2->4->6->7->10->	PointCircle
73	DE	1->2->4->6->8->9->10->	PointCircle
74	DE	1->2->3->10->	PointCircle
75	DE	1->2->4->5->10->	PointCircle
76	DE	1->2->4->6->7->10->	PointCircle

id	algo	path	program
77	Bacterial	1->2->4->6->8->9->10->	Quadrant
78	Bacterial	1->2->3->10->	Quadrant
79	Bacterial	1->2->4->5->10->	Quadrant
80	Bacterial	1->2->4->6->7->10->	Quadrant
81	DE	1->2->4->6->8->9->10->	Quadrant
82	DE	1->2->4->5->10->	Quadrant
83	DE	1->2->3->10->	Quadrant
84	DE	1->2->4->6->7->10->	Quadrant

id	algo	path	program
89	Bacterial	1->2->3->5->7->8->10->12->13->14->15->24->	Traingle Classifier
90	Bacterial	1->2->16->17->19->21->22->23->24->	Traingle Classifier
91	Bacterial	1->2->3->5->7->8->9->14->15->24->	Traingle Classifier
92	Bacterial	1->2->16->17->19->20->23->24->	Traingle Classifier
93	Bacterial	1->2->3->4->15->24->	Traingle Classifier
94	Bacterial	1->2->16->17->18->23->24->	Traingle Classifier
95	Bacterial	1->2->3->5->6->15->24->	Traingle Classifier
96	DE	1->2->3->4->15->24->	Traingle Classifier
97	DE	1->2->16->17->19->21->22->23->24->	Traingle Classifier
98	DE	1->2->3->5->7->8->10->11->14->15->24->	Traingle Classifier
99	DE	1->2->16->17->18->23->24->	Traingle Classifier
100	DE	1->2->16->17->19->20->23->24->	Traingle Classifier
101	DE	1->2->3->5->7->8->9->14->15->24->	Traingle Classifier
102	DE	1->2->3->5->6->15->24->	Traingle Classifier
103	DE	1->2->3->5->7->8->10->12->13->14->15->24->	Traingle Classifier

id	algo	path	program
109	Bacterial	1->2->3->10->	Marks
110	Bacterial	1->2->4->6->7->10->	Marks
111	Bacterial	1->2->4->6->8->9->10->	Marks
112	Bacterial	1->2->4->5->10->	Marks
113	DE	1->2->4->6->8->9->10->	Marks
114	DE	1->2->3->10->	Marks
115	DE	1->2->4->5->10->	Marks
116	DE	1->2->4->6->7->10->	Marks

id	algo	testcase	program
171	Bacterial	[1] (13,5,5,0,);	Traingle Classifier
172	Bacterial	[2] (2,12,6,6,);	Traingle Classifier
173	Bacterial	[3] (2,2,9,2,);	Traingle Classifier
174	Bacterial	[4] (9,12,18,3,);	Traingle Classifier
175	Bacterial	[5] (2,19,20,3,);	Traingle Classifier
176	Bacterial	[6] (4,13,12,3,);	Traingle Classifier
177	Bacterial	[7] (10,13,1,6,);	Traingle Classifier
178	Bacterial	[8] (13,5,9,3,);	Traingle Classifier
179	Bacterial	[9] (13,12,6,3,);	Traingle Classifier
180	Bacterial	[10] (10,5,20,2,);	Traingle Classifier
181	Bacterial	[11] (10,12,6,3,);	Traingle Classifier
182	Bacterial	[12] (13,19,9,3,);	Traingle Classifier
183	Bacterial	[13] (2,13,9,6,);	Traingle Classifier
184	Bacterial	[14] (13,13,5,1,);	Traingle Classifier
185	Bacterial	[15] (9,13,6,3,);	Traingle Classifier
186	Bacterial	[16] (9,5,6,3,);	Traingle Classifier
187	Bacterial	[17] (10,19,20,3,);	Traingle Classifier
188	Bacterial	[18] (13,2,5,4,);	Traingle Classifier
189	Bacterial	[19] (2,5,9,2,);	Traingle Classifier
190	Bacterial	[20] (4,12,6,6,);	Traingle Classifier
191	Bacterial	[21] (10,2,5,4,);	Traingle Classifier
192	Bacterial	[22] (4,2,6,2,);	Traingle Classifier
193	Bacterial	[23] (4,5,5,7,);	Traingle Classifier
194	Bacterial	[24] (4,19,5,6,);	Traingle Classifier

id	algo	testcase	program
142	Bacterial	[1] (3,1,17,0,);	EvenOdd
143	Bacterial	[2] (19,1,9,2,);	EvenOdd
144	Bacterial	[3] (8,2,15,3,);	EvenOdd
145	Bacterial	[4] (19,7,3,2,);	EvenOdd
146	Bacterial	[5] (3,1,17,2,);	EvenOdd
147	Bacterial	[6] (1,2,9,1,);	EvenOdd

id	algo	testcase	program
97	DE	[1] (2,13,3,0,);	MaximumofThree
98	DE	[2] (15,18,1,3,);	MaximumofThree
99	DE	[3] (19,3,7,2,);	MaximumofThree
100	DE	[4] (15,9,4,2,);	MaximumofThree
101	DE	[5] (2,13,7,3,);	MaximumofThree
102	DE	[6] (9,7,17,1,);	MaximumofThree
103	DE	[7] (2,3,3,3,);	MaximumofThree

id	algo	testcase	program
108	DE	[1] (7,10,5,0,);	EvenOdd
109	DE	[2] (9,19,13,1,);	EvenOdd
110	DE	[3] (13,3,5,1,);	EvenOdd
111	DE	[4] (7,10,5,1,);	EvenOdd
112	DE	[5] (20,18,20,2,);	EvenOdd
113	DE	[6] (4,19,4,2,);	EvenOdd
114	DE	[7] (13,11,12,1,);	EvenOdd
115	DE	[8] (20,10,5,2,);	EvenOdd
116	DE	[9] (17,8,17,1,);	EvenOdd
117	DE	[10] (2,16,2,2,);	EvenOdd
118	DE	[11] (7,8,5,1,);	EvenOdd
119	DE	[12] (7,18,5,1,);	EvenOdd
120	DE	[13] (19,14,16,1,);	EvenOdd
121	DE	[14] (7,16,5,1,);	EvenOdd
122	DE	[15] (5,11,18,1,);	EvenOdd
123	DE	[16] (9,10,20,1,);	EvenOdd
124	DE	[17] (3,4,5,1,);	EvenOdd
125	DE	[18] (2,19,20,2,);	EvenOdd
126	DE	[19] (8,15,13,2,);	EvenOdd
127	DE	[20] (13,19,5,1,);	EvenOdd
128	DE	[21] (12,11,15,2,);	EvenOdd
129	DE	[22] (6,5,12,2,);	EvenOdd
130	DE	[23] (3,17,17,1,);	EvenOdd
131	DE	[24] (2,9,10,2,);	EvenOdd
132	DE	[25] (13,10,5,1,);	EvenOdd
133	DE	[26] (12,15,15,2,);	EvenOdd
134	DE	[27] (6,10,5,2,);	EvenOdd
135	DE	[28] (4,1,2,2,);	EvenOdd

id	algo	testcase	program
104	DE	[1] (12,0,);	Leap Year
105	DE	[2] (19,2,);	Leap Year
106	DE	[3] (34,2,);	Leap Year
107	DE	[4] (12,1,);	Leap Year

id	algo	testcase	program
92	DE	[1] (5,19,0,);	MinofTwo
93	DE	[2] (9,18,2,);	MinofTwo
94	DE	[3] (5,19,2,);	MinofTwo
95	DE	[4] (9,19,2,);	MinofTwo
96	DE	[5] (5,18,2,);	MinofTwo

id	algo	testcase	program
1	DE	[1] (18,18,0,);	Marks
2	DE	[2] (13,2,2,);	Marks
3	DE	[3] (6,9,2,);	Marks
4	DE	[4] (18,13,2,);	Marks
5	DE	[5] (13,18,2,);	Marks
6	DE	[6] (13,9,2,);	Marks
7	DE	[7] (18,2,2,);	Marks
8	DE	[8] (13,13,2,);	Marks
9	DE	[9] (6,13,2,);	Marks
10	DE	[10] (18,9,2,);	Marks
11	DE	[11] (18,18,2,);	Marks
12	DE	[12] (6,2,2,);	Marks
13	DE	[13] (6,18,2,);	Marks

id	algo	testcase	program
79	DE	[1] (18,3,0,);	PointCircle
80	DE	[2] (15,7,1,);	PointCircle
81	DE	[3] (15,12,1,);	PointCircle
82	DE	[4] (5,14,1,);	PointCircle
83	DE	[5] (5,3,1,);	PointCircle
84	DE	[6] (15,3,1,);	PointCircle
85	DE	[7] (5,7,1,);	PointCircle
86	DE	[8] (18,3,1,);	PointCircle
87	DE	[9] (18,7,1,);	PointCircle
88	DE	[10] (5,12,1,);	PointCircle
89	DE	[11] (18,12,1,);	PointCircle
90	DE	[12] (18,14,1,);	PointCircle
91	DE	[13] (15,14,1,);	PointCircle

id	algo	testcase	program
19	DE	[1] (9,18,15,0,);	Traingle Classifier
20	DE	[2] (4,3,3,3,);	Traingle Classifier
21	DE	[3] (13,7,16,6,);	Traingle Classifier
22	DE	[4] (2,4,5,6,);	Traingle Classifier
23	DE	[5] (15,14,1,5,);	Traingle Classifier
24	DE	[6] (5,7,6,6,);	Traingle Classifier
25	DE	[7] (1,18,16,2,);	Traingle Classifier
26	DE	[8] (18,9,13,6,);	Traingle Classifier
27	DE	[9] (9,18,15,6,);	Traingle Classifier
28	DE	[10] (13,3,1,5,);	Traingle Classifier
29	DE	[11] (9,9,16,7,);	Traingle Classifier
30	DE	[12] (12,13,4,6,);	Traingle Classifier
31	DE	[13] (5,18,15,6,);	Traingle Classifier
32	DE	[14] (5,3,3,3,);	Traingle Classifier
33	DE	[15] (3,7,6,6,);	Traingle Classifier
34	DE	[16] (9,14,15,6,);	Traingle Classifier
35	DE	[17] (18,13,3,5,);	Traingle Classifier
36	DE	[18] (19,9,15,6,);	Traingle Classifier
37	DE	[19] (16,14,3,6,);	Traingle Classifier
38	DE	[20] (19,3,15,5,);	Traingle Classifier
39	DE	[21] (4,9,3,2,);	Traingle Classifier
40	DE	[22] (9,3,6,5,);	Traingle Classifier
41	DE	[23] (17,7,9,5,);	Traingle Classifier
42	DE	[24] (18,7,3,5,);	Traingle Classifier
43	DE	[25] (18,18,15,7,);	Traingle Classifier
44	DE	[26] (18,8,16,6,);	Traingle Classifier
45	DE	[27] (6,4,6,1,);	Traingle Classifier
46	DE	[28] (13,18,15,6,);	Traingle Classifier

4.1.1 Implementation Issues

While creating the 0-1 matrix, there might be a possibility than all the values in a given row are less than the threshold value decided for Differential Evolution Algorithm. To resolve such an issue, threshold value was decremented by a margin of 0.1 and then the algorithm was rerun on the same row. If the problem still exist, It means if there is still no such value in the whole row which is greater than the new threshold value, we further reduce the threshold value until we find a solution. In the worst case, we are guaranteed to find our solution when the threshold value becomes zero. Threshold is set to its initial value for further computation for 0-1 matrix. Furthermore, run time of algorithm increases rapidly with an increase in number of test cases and hence the system becomes slower and slower. This is due to the amount of permutations and combinations that are performed in the Differential Evolution algorithm.

4.1.2 Algorithms

A. PSEUDO CODE FOR DE

BEGIN
  Initial population is randomly generated.
  Fitness value is calculated for each parent present in initial population.

REPEAT
  For each parent (X) test case select three random children from the population (g,h,i) each distinct from each other and as well as the parent.
  Create one new test suite using the DE FUNCTION.
  Do this step equal to the population size.
  For each new sibling check
    if it has more fitness value than parent , then replace the parent with the improved variant in the initial population.
    else leave the parent as it is.
Until stopping condition is not met.

B. PSEUDO CODE FOR DE FUNCTION

a. Randomly pick an index $RI \in (1\ldots N)$.
b. Compute the new sibling's new position $P=\{P_1,P_2,\ldots, P_N\}$ by:
   (i) For each $k \in \{1\ldots N\}$ pick uniformly distributed number such that $u_k \cong (0,1)$.
   (ii) If (u_k< CP) or (k = RI) then
   $$P_k = g_k + D\_W *(h_k - i_k)$$
   else $P_k = X_k$
   (iii) If (fit (P_k) > fit (X_k)) then

Replace the parent in population with the improved variant in the population.z

Here, D_W is differential weight ∈ (0,2) and CR is crossover probability∈ (0,1) both are selected by the user.

4.2 Risk Analysis and Mitigation Plan

Risk Id	Description of Risk	Probability (P)	Impact (I)	RE (P*I)	Risk selected for mitigation	Contingency Plan
1	The user may not be able to understand the interface.	L	H	M	Y	N
2	The technology used to develop this product is not known by team members.	M	M	M	N	Y
3	The system running the Product does not have the minimum requirement specifications.	L	H	H	Y	N
4	The NP hard Problem given to the product was not optimized upto the standards.	L	M	M	N	Y
5	The software required for the product are not present in the system	H	H	H	Y	N

Table 2 (Risk Analysis)

Mitigation Plan-

Risk Id	Proposed plan
1	The product's front end will be designed in such a way that it will be very basic and easy to understand moreover a step by step guidance will be provided and a demo would be given in order to help the user to start the work.
2	The Products minimum system requirement will be checked and then only the user will be able to go further in the product if its not meant then we will request the user to upgrade his system hardware in order to match the requirements or maybe in future we will launch a lite version for it.
3	The software used in the project are free source and can be downloaded form the internet easily. Moreover a requirement list for the product will also be given to the user along with the product.

Table 3 (Mitigation Plan)

Contingency Plan-

Risk Id	Proposed plan
1	The technology used for the development is JAVA and DBMS and is quite simple and the team members can be easily taught by Training and regular
2	The NP hard problem given to the Product will not be optimized upto the standards but some optimization will surely be there.

Table 4 (Contingency Plan)

Chapter 5 - Testing

5.1 Testing Plan

Activity	Start Date	Completion Date	Hours	Comments
Build Test plan	15/01/17	20/01/17	5	The purpose of the Test Plan deliverable is to define a detailed, comprehensive plan for controlling and testing the application by using unit test cases and integration test cases by black box or white box testing technique.
Approach Test	28/01/17	29/01/17	2	The Test Approach describe the sources project documentation for requirements that will be used to drive the test design. This includes the modeling outputs (e.g., outputs from the process modeling activities such as use cases and task scenarios) from which test cases are developed. and execution
Test Strategy	17/02/17 & 24/04/17	20/02/17 & 26/04/17	3 + 4	The Test Strategy defines overall approach for ensuring the quality of the application prototype. Specifically, the test strategy describes the scope, levels, objectives, completion criteria, and estimated resources for testing activities.
Unit Testing	28/02/17 & 30/04/17	04/03/17 & 03/05/17	3 + 3	Testing is done many times for a small piece of source code.
Integration Testing	10/03/17	13/03/17	5	It is a combination of many unit test cases.
Testing Environment	24/03/17 & 06/05/17	25/03/17 & 07/05/17	2 + 3	The definition of the technical environment used for the testing of a computer application.
Stress, Load, Volume	20/04/17	20/04/27	2	These are used when we do entry in database and depends on how much we entered in database.
Testing Environment	12/03/17 & 10/05/17	14/03/17 & 11/05/17	4 + 3	The definition of the technical environment used for the testing of a computer application.

Table 5 (Testing Plan)

5.2 Component decomposition and type of testing required

S.No	Module for testing	Type of Testing Required	Technique for writing test cases
1.	Input array	Unit testing	Black Box
2.	Input test cases	Unit testing	Black Box
3.	Entry in database	Volume testing, Stress testing, Load testing	Black Box
4.	Branch coverage	Unit testing	White Box
5.	Fault coverage	Unit testing	White Box
6.	General Feature Selection	Unit testing	Black Box
7.	Fitness function	Integration testing	White Box

Table 6 (Type Of Testing Required)

Test cases for component input array module

Equivalence Classes is integer and Boundary Value classes is [1,100].

Test Case id	Input	Expected Output	Status
1.	{0}{0}	ERROR	Pass
2.	{-7,-6,-20,-4}{4}	Test input Module begin	Pass
3.	{2,7,-8,-9}{4}	Test input Module begin	Pass

Table 7 (Testing Input Array Module)

Test cases for component input test cases module

Equivalence Classes is integer and Boundary Value classes is [1, 20].

Test Case id	Input	Expected Output	Status
1.	{0,6,-16}	Entered in database	Pass
2.	{0,6,-1}	Entered in database	Pass
3.	{0,7,14}	Entered in database	Pass

Table 8 (Testing Input Test Case Module)

Test cases for component entry in databases module

Equivalence Classes is integer and Boundary Value classes is [1, 20].

Test Case id	Input	Expected Output	Status
1.	{0,6,-16}	Stored in Database	Pass
2.	{0,6,-8}	Stored in Database	Pass
3.	{0,7,14}	Stored in Database	Pass

Table 9 (Testing Component Entry)

Test cases for component entry in branch coverage module

Equivalence Classes is integer and Boundary Value classes is [1, 20].

Test Case id	Input	Expected Output	Status
1.	NA	{Y,N,Y,N}	Pass
2.	NA	{Y,Y,Y,Y}	Pass
3.	NA	{Y,N,N,Y}	Pass

Table 10 (Branch Coverage Module)

Test cases for component entry in fault coverage module

Equivalence Classes is integer and Boundary Value classes is [1, 20].

Test Case id	Input	Expected Output	Status
1.	NA	{Y,Y,N,Y,N}	Pass
2.	NA	{Y,Y,Y,Y,Y}	Pass
3.	NA	{Y,Y,N,N,Y}	Pass

Table 11 (Fault Coverage Module)

Test cases for component entry in general feature selection module

Equivalence Classes is between 0 to1 which is decimal. Since, we are taking 20x20 matrices. So, Boundary Value classes is [1, 400].

Input

0.12003400367498596	0.26741628955603325	0.0012561649851646761	0.3294581458181707	0.18101120136072846

Table 12 (Random Row)

2. Random matrix (automatic generated randomly)

0.3195145613401611	0.9816107371507137	0.07842108918201818	0.4796994138629581	0.0124664778098552
0.7564077516575008	0.4441955113580993	0.589594899723035	0.7263080415531624	0.14128961010870367
0.6526835466581318	0.32826832661517513	0.02377581625746561	0.9329759939407378	0.3188718295881934
0.3510491056741064	0.5069054487671629	0.9315185036615858	0.4432007828906608	0.6629542824086172
0.7948543062795658	0.025590355906243034	0.39623920425822257	0.9636871798390512	0.6327449950891065

Table 13 (Random Matrix)

Output

1.0	1.0	0.0	0.0	0.0
1.0	0.0	0.0	1.0	0.0
0.0	1.0	1.0	0.0	0.0
1.0	0.0	1.0	0.0	0.0

Table 14 (General Feature Matrix)

Test cases for component entry in fitness function module

Equivalence Classes is integer and Boundary Value classes is [1, 20!].

Test Case id	Input	Expected Output	Status
1.	{0,6,-16}{0,6,-8}	100%	Pass
2.	{0,6,-16}{0,7,14}	80%	Pass
3.	{0,6,-8}{0,7,-14}	100%	Pass

Table 15 (Fitness Module Test Cases)

5.3 List all test cases in prescribed format

Type of Test	Will Test be Performed	Comments and Explanations	Software Components
Requirement Testing	YES	It is the process of gathering requirements, which are shown below. So, here if test cases are not generated the project will be of no use. As, this testing helps in creating test cases and executing them.	Netbeans, jdk
Unit	YES	A unit is the smallest testable piece of software, which has been used regularly in our project at each step like input array and test cases etc.	Netbeans, jdk, junit
Integration	YES	A software program may have many unit test cases. It is used in to calculate fitness function because here we have to combine both branch and fault coverage which are unit testing.	Netbeans, jdk, junit
Performance	NO	Execution testing is the procedure of hinder mining the speed or viability of a computer, arrange, programming project or gadget. We felt no need of utilizing it.	Load Runner
Stress	YES	It involves execution of application with more than maximum and varying loads. It is used when we do entry in database and depends on how much we entered in database.	Jmeter
Compliance	NO	Consistence testing, is a philosophy utilized as a part of designing to guarantee that an item, handle, PC program or framework meets a characterized set of guidelines.	Ada
Security	NO	Security is the procedure used to protect information from various threats. So, there is no need for any privacy and there is no information about anything.	Nmap
Load	YES	Load testing involves testing the application or system to behave under normal and at a peak time and measure it response. It is used in database and depends on how much entry is entered in database.	WebLoad, Load runner

Volume	YES	Volume testing alludes to testing a product application or the item with a specific measure of information. Along these lines, It is utilized when we do section in database.	Jmeter
System Testing	YES	It is a combination of the software, hardware and other associated parts that together provide product solutions. It is based on risk, requirement etc.	Netbeans, JDK

Table 16 (Test Cases In Prescribed Form)

5.4 Limitations of the Solution

Software Requirements

The product is developed on Netbeans and Wamp server and requires the knowledge of JAVA and DBMS. The requirement for these Software are as follows:-

1. NetBeans bundles only require the Java Runtime Environment (JRE) 7 or JRE 8 to be installed and run.
2. Java features in the IDE require JDK 7 or JDK 8. JavaFX 2.2 (or newer) features required.

Hardware Requirements

1. Microsoft Windows XP Professional SP3/Vista SP1/Windows 7 Professional:
- ◦ Processor: 800MHz Intel Pentium III or equivalent
- ◦ Memory: 512 MB
- ◦ Disk space: 750 MB of free disk space

2. Ubuntu 9.10:
- ◦ Processor: 800MHz Intel Pentium III or equivalent
- ◦ Memory: 512 MB
- ◦ Disk space: 650 MB of free disk space

3. Macintosh OS X 10.7 Intel:
- ◦ Processor: Dual-Core Intel
- ◦ Memory: 2 GB
- ◦ Disk space: 650 MB of free disk space

6. Findings & Conclusion

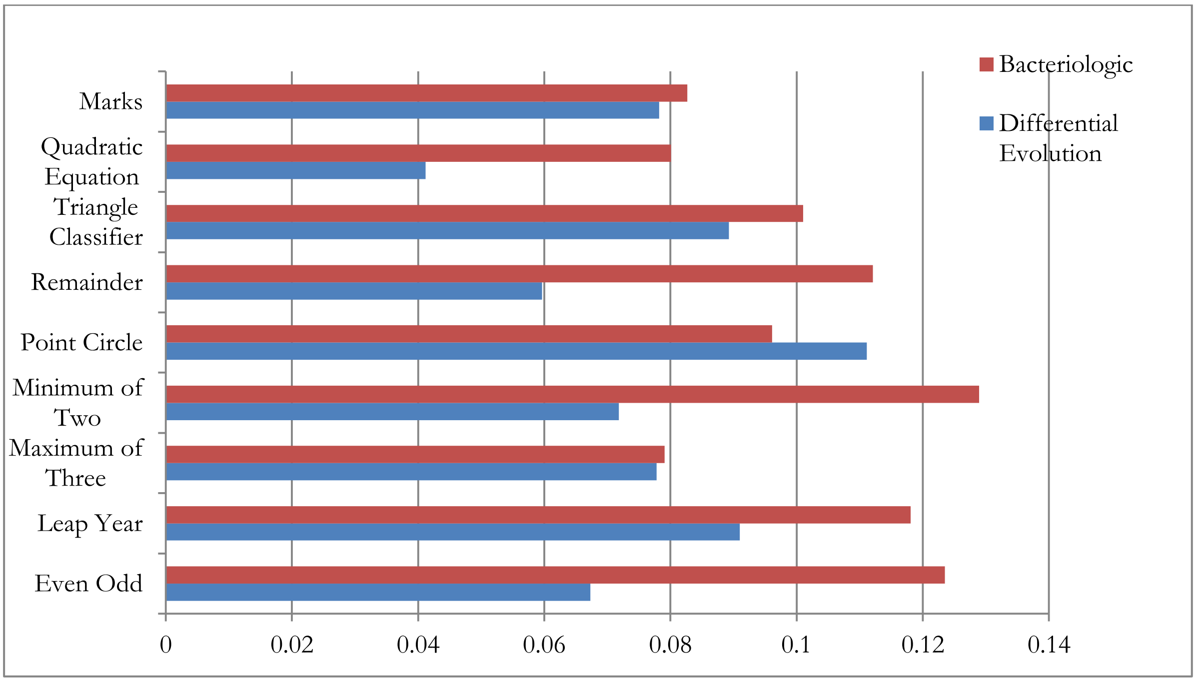
6.1 Findings

Here ,we find out that both Bacteriologic and Differential evolution being black box algorithms gives us the good results by choosing the optimal test case, and further optimizing and prioritization them to make better out of them. Moreover, they both were quite successfully prioritizing test cases and computed them very fast.
Genetic algorithm is very popular and used and various technologies and tools and basic operations of genetic are selection, crossover and mutation.
Differential evolution is actually a specific subset of the broader space of genetic algorithms, with some restrictions. The three evolutionary operations are involved in basic forms of differential evolution: Mutation, Recombination and Selection.

Populations are initialized randomly for both the algorithms between upper and lower bounds of the respective decision space. Size of the population should be at least four in the differential evolution and two for genetic algorithm. New population will be created in every generation, which will be performed by some evolutionary operations sequentially. Until a termination criterion is met, such generations will be repeated.

6.2 Conclusion

Differential Evolution algorithm is finding the true global minimum of a multi modal search space regardless of the initial parameter values, fast convergence, and using a few control parameters. Differential Evolution algorithm and Bacteriologic Algorithm both are population based but does not guarantee, optimization even for non-differentiable, non-continuous objectives as both are prone to the fact of getting struck at local maxima and local minima. Therefore sometimes they are not able to give the optimum result.



6.3 Future Works

We wanted to compare many inspired algorithms with each other like bacteriologic, differential, Grey wolf optimization. Moreover we wanted to write research paper on grey wolf optimization and Differential Evolution Technique for Test-Case Optimization (DETCO) which would focus on the topic of test case optimization by enhancing the most out of Bacteriologic and Differential algorithm.

References

1. Rothermel G., Untch R.J. and Chu C. 'Prioritizing test cases for regression testing', Vol. 27,No. 10, (2001).
2. Becerra R.L., Sagarna R. and Yao X., Evaluation of Differential Evolution in Software Test Data Generation.
3. Jeffrey, D. and Gupta, N., ‘Improving fault detection capability by selectively retaining test cases during test suite reduction’, Vol. 33, No. 2.
4. Rothermal G., Harrold M.J., Ostrin J. and Hong C. ‘An empirical study of the effects of minimization on the fault detection capabilities of test suites’, (1998).
5. Srivastava P.R.-“Test case optimisation a nature inspired approach using bacteriologic algorithm “, Vol. 8, No. 2, 2016.
6. Tanenbaum A.M. Data Structures using C and C++, 2nd edition(2008).
7. Aluffi-Pentini F., Parisi V. and Zirilli F. Global Optimization and Stochastic Differential Equations, Journal of Optimization Theory and Applications47(1), (1985).
8. Pressman, R.S. "Software Engineering: A Practitioner’s Approach", 5th ed(2001).
9. Cormen T.H., Leiserson C.E. and Rivest R.L. Introduction to Algorithms(2001).
10. Wong W.E., Horgan J.R., London S. and Mathur A.P.‘Effect of a test set minimization on fault detection effectiveness’,(1995).
11. Holland J.H. Adaptation in Nature and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence(1992).
12. Krishnamoorthi R. and Mary S.A.S.A.‘Regression test suite prioritization using genetic algorithms’, Vol. 2, No. 3(2009).
13. Baudry B., Fleurey F., Jézéquel J-M. and Le Traon Y.‘Automatic test case optimization: a bacteriologic algorithm’, Vol. 22, No. 2, (2005).
14. Price K.; Storn, R.M.; Lampinen J.A. Differential Evolution: A Practical Approach to Global Optimization. Springer. ISBN 978-3-540-20950-8(2005).

Brief Resume of Students-

Vibhor Gupta
Male, 21 yrs, New Delhi
Vibhor.gupta9913103515@gmail.com
Vibhor.gupta1110@gmail.com
+91-9999959254

Job objective
To secure a position in your organisation that would make use of and entail development of all my skills and knowledge and whereby I can contribute to the success of the organisation.
Academic Qualifications:

Degree/Certificate	Specialization/Minor	Board/University	Year	Percent-age/SGPA
B.Tech (6 sem avg.)	CSE	Jaypee Institute of Information Technolo-gy,Noida	2013-16	7.8
HSC	Science	CBSE,Happy School	2013	89.4%
SSC	---	CBSE,Happy School	2011	9.2

Projects:

1. **Title:** FINDEN
   Type: Website
   Position: Developer (Group member)
   Language and others: HTML+CSS, PHP, JavaScript, SQL
   Summary: A website to locate the nearby shops as well as show their inventories and rate list on the basis of your shopping list item(s).

2. **Title:** LIFE CONNECT
   Type: Android Application
   Position: Developer (Group member)
   Language and others: JAVA, XML, SQLlite
   Summary: An android application focusing on the improvement of medical sector. Mainly focused on 4 sections: - Ambulance calling and tracking, Connection Blood donors and needy directly, Organizing Blood Donation camps, Discussion thread. It used location, camera and other hardware and services.

Other projects:

1. **Title:** Mailbox
   Position: Developer (Group member)
   Language and others: JAVA, Oracle

Summer Training:

1. **Summer Training at Xperia technologies Pvt. Ltd.**
   Duration: 6 weeks
   Type: Android Application
   Position: Junior Developer

Technical Expertise:

- **PROGRAMMING:** C++, C
- **MOBILE DEVELOPMENT:** ANDROID
- **WEB DEVELOPMENT:** HTML + CSS
- **DATABASE:** SQL

Achievement:

- Gold medal winner in international information Olympiad at school level.
- House captain at school level.

Strengths:

- Quick learner
- Optimistic
- Strong motivational and leadership skills.

NAMAN SHARMA
C-134 Anand Vihar, Delhi – 110092
Email: naman3546@gmail.com
Phone: +91-8527072442

OBJECTIVE

To associate with a vibrant organization, to fully utilize my knowledge, skills and serve the organization well and keep learning and improving.

EDUCATIONAL QUALIFICATION

B.Tech in Computer Science Engineering from JIIT, Noida with 6.5 CGPA (2013-present).
HSC from CBSE Board with 86% Marks.
SSC from CBSE Board with 9.0 CGPA.

TECHNICAL SKILLS

Programming Languages: C, JAVA (Core, Servlets, JSP)
Software: Netbeans, Eclipse, Adobe Photoshop
Database: MySQL, Oracle
Familiar With: Data Structures

WORK EXPERIENCE

- 6 week summer internship (13 June – 29 July, 2016) from Department of Electronics & Information Technology (DEITY). Authenticating and improving Digital signatures by algorithms using C, JAVA, Data Structures.
- 6 week summer training in JAVA. Developed a mailbox as a project.

PROJECTS COMPLETED

- Prasedium (Android Application) in JAVA, Android as 2nd Minor Project in 2016.
- Translator Exegete (Android Application) in JAVA, Android as 1st Minor Project in 2015.
- QuizMania (Software) using Core Java, Swings API and Oracle as Database Project in 2014.
- MailBox (Software) using Core Java, Swings API and Oracle as OOPS Project in 2014.

ACHIEVEMENTS

- Part of Cricket team for tournament in LNMIIT, Jaipur in 2014.
- Part of Coordinating team for Cricket competition at JIIT-128 annual fest, Converge in 2016.
- Part of creative team for JIIT-128 annual fest, Converge in 2015.
- Participated in 3 day technical event, DevFest 2015 conducted by Google Developers Group, JIIT Noida.

KEY SKILLS AND STRENGTHS

Quick Learner, Good analytical skills, Focused and punctual, Good grasping ability, Friendly, Honest.

AVNEET SINGH
+91-9958164550
Singhavneet77@gmail.com

CAREER OBJECTIVE

To work in an established organization and help in contributing to the growth of the esteemed company and to stand up to the company's expectations with honesty and dedication.

SKILLS & STRENGTHS

1. Flexibility and Adaptability
2. Quick learner
3. Good interpersonal skills
4. Trained in C++, Java Core and Android.
5. Good Communication skills
6. Good team player

EDUCATION

•	2013 - Class 12 from S.S Mota Singh sr. sec model school , New Delhi	**78.2**
•	2011- Class 10 from S.S Mota Singh sr. sec model school , New Delhi	**81.7**

PROJECTS AND WORK EXPERIENCE

Public utility finder android application which helps in finding college, school, hospitals nearby and paying electricity and water bills through app.
Distribution of file system using cryptography
I had worked as a summer trainee for 6 months at Tech Vision IT under guidance of HCL ,Delhi on an Android Application development.

HOBBIES & INTERESTS

Dancing, singing, football.

PERSONAL DETAILS	
Address	VB -155 Varinder Nagar, Street No. 7 New Delhi-110058
Date of Birth	13 November,1994
Languages	English, Hindi, Punjabi