-

Test Case Reduction Using Genetic Algorithm

A Mini Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Bachelor of Technology
in
Computer Science & Engineering

by

ANIL BALCHANDANI (20144055)
ABHINANDAN AGARWAL (20144091)
JITESH (20144123)
DHEERENDRA KUMAR(20144133)
MANOJ KUMAR (20144155)

to the

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
MOTILAL NEHRU NATIONAL INSTITUTE OF TECHNOLOGY
ALLAHABAD,UP,INDIA(211004)
April, 2017

UNDERTAKING

We declare that the work presented in this report titled "***Test Case Reduction Using Genetic Algorithm***", submitted to the Computer Science and Engineering Department, Motilal Nehru National Institute of Technology Allahabad, for the award of the ***Bachelor of Technology*** degree in ***Computer Science & Engineering***, is our original work. I have not plagiarized or submitted the same work for the award of any other degree. In case this undertaking is found incorrect, I accept that my degree may be unconditionally withdrawn.

April, 2017
Allahabad

ANIL BALCHANDANI
(20144055)
ABHINANDAN AGARWAL
(20144091)
JITESH (20144123)
DHEERENDRA
KUMAR(20144133)
MANOJ KUMAR (20144155)

CERTIFICATE

Certified that the work contained in the report titled "***Test Case Reduction Using Genetic Algorithm***", by *ANIL BALCHANDANI(20144055) , ABHINANDAN AGARWAL(20144091) , JITESH(20144123), DHEERENDRA KUMAR(20144133) , MANOJ KUMAR(20144155)*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Dr. Anoj Kumar)
Computer Science and Engineering Dept.
M.N.N.I.T Allahabad

April, 2017

Preface

Software Testing is one of the crucial part of software development life cycle that is being carried out thoroughly. The time and cost required for the testing are the major issues in the present scenario.

We are trying to develop a method to remove the redundant test cases from the test suite so as to speed up the process of testing. It becomes useful, when a software is developed in parts and then integrated together, while integrating each module it is to be tested again with some new set of test cases along with the old ones. With reduction in the number of test cases used (through coverage criteria) the testing process can become fast and effective.

Since the intermediate problems are NP-complete, we will be using optimisation algorithms like genetic algorithm for reaching optimality in our results.

Acknowledgements

We would like to express our deepest appreciation to all those who provided us the possibility to complete this report. A special gratitude we give to our project mentor,**Dr.Anoj Kumar** whose contribution in stimulating suggestions and encouragement, helped us to coordinate our project.Then we would like to thank all our friends, colleagues for their support and motivation.Last and the most important, we would like to thank God Almighty for his constant blessings and love.

Also, we would like to thank the director of our institute **Prof. Rajeev Tripathi** for his contribution and support towards the students of this institute and also we would like to thank our head of the department(HOD) **Prof. Neeraj Tyagi** who has shown a kind of interest in such activities for the development of students.In the end we are thankful to all those who are directly or indirectly have helped and directed us at every possible step.

Contents

Preface **iv**

Acknowledgements **v**

1 Abstract **1**

2 Introduction **2**

2.1 Motivation 2

2.2 Software Development Model Used 3

3 Overview **4**

3.1 Test Case Reduction and Related Work 4

3.2 Genetic Algorithm 5

3.3 Problem Statement 7

3.4 Framework 8

4 Proposed Approach **14**

5 Experimental Setup And Results **20**

5.1 Dataset Used 20

5.2 Tools and System Used 20

5.3 Work done and Results obtained 21

6 Conclusions **28**

7 Future Work **29**

References **30**

Chapter 1

Abstract

Software testing is one of the important stages of software development. In software development, developers always depend on testing to reveal bugs. In the maintenance stage test suite size grow because of integration of new technique. Addition of new technique force to create new test case which increase the size of test suite. In regression testing new test case may be added to the test suite during the whole testing process. These additions of test cases create possibility of presence of redundant test cases. Due to limitation of time and resource, reduction techniques should be used to identify and remove them. Research shows that a subset of the test case in a suit may still satisfy all the test objectives which is called as representative set. Redundant test case increase the execution cost of the test suite, in spite of NP-completeness of the problem there are few good reduction techniques have been available. In this paper a new approach for test case reduction is proposed. This algorithm use genetic algorithm technique iteratively with varying chromosome length to reduce test case in a test suit by finding representative set of test cases that are fulfill the testing criteria.

Chapter 2

Introduction

Retesting of software is done frequently during the software development life cycle and in particular in regression testing. In regression testing software grows and evolves that create new test cases and added them to a test suite to exercise the latest changes in the software[5]. Due to many versions of the development of the projects, the possibility of redundant test cases in test suite is more .The redundant test case may in respect to the testing requirements for which they were generated. Due to limitation of time and resource for retesting the software every time before a new version is released, it is really important to search for techniques that ensure manageable test suites size by removing redundant test cases without hampering the performance of the software. This process is popularly called test suite minimization.

2.1 Motivation

The motivation behind this topic is that the time taken and cost involved in testing phase of software development life cycle increases rapidly as the test suite size increases.So to obtain the correct results i.e. minimized test suite that will reduce the cost and effort involved in testing.

2.2 Software Development Model Used

We have used incremental model of software development life cycle as after the Requirement Gathering and Analysis we have built different modules and each module is being tested before we built the next module to reach at the final results.

Chapter 3

Overview

3.1 Test Case Reduction and Related Work

As the software is modified and new test cases are added to the test- suite, the size of the test-suite grows and the cost of regression testing increases.This paper investigates the use of an evolutionary approach, called genetic algorithms, for test-suite reduction[2]. The algorithm builds the initial population based on test history, calculates the fitness value using coverage and cost information, and then selectively breeds the successive generations using genetic operations. This generational process is repeated until a minimized test-suite is found[4].

Test-suite reduction techniques have been extensively studied. Harrold, Gupta, and Soffa proposed a methodology for controlling the size of a test suite. Jones and Harrold presented an algorithm for test-suite reduction that can be tailored effectively for use with Modified Condition/Decision Coverage (MC/DC)[1].

In order to perform test-suite reduction, we should do something includes:

- Maintaining a testing pool where contains all the test cases used in previous test activities.

- Keeping the test coverage information which denotes how many and which parts of the program tested by each test cases during the previous tests.
- Recording the test-execution cost information that measures the amount of resources each test-cases execution needs.

.

A test-suite reduction technique should possess several qualifications listed in the following.

- Adequacy, which means that the reduced test-suite must provide the same test coverage of the software, according to some criterion, as the original test-suite.
- Precision, which means that the test-suite algorithm must be able to reduce the redundant test cases from the original test-suite, that is, be able to find a minimal or an approximate minimal subset of test cases.
- Cost-effectiveness, which means that it is worth doing test-suite reduction only if the cost of the analysis necessary to do test-suite reduction is less than the savings realized by reducing the test-suite.
- Generality, which means the technique must be general, that is, it must be applicable to a wide class of program and modification.

3.2 Genetic Algorithm

Genetic algorithm is population-based search algorithm inspired from principles of natural evolution called as evolutionary algorithm (EA) [6] . These algorithms are general-purpose optimization algorithms with probabilistic components .Originally, GA was developed to optimally solve sequential decision processes but over the years, it has been used in learning as well as optimization problem. GA works with a population of points instead of

working with a single point.Each point is considered as vector in hyperspace. Each vector is called a chromosome which is represented as a binary string. Number of elements in each chromosome is same as the number of parameters in optimization problem[3]. A general series of operations carried out while applying a GA are as follows:

- Initialize the population.
- Calculate fitness for each chromosome in population.
- Reproduce the selected chromosome to form a new population.
- Apply crossover and mutation on the population.
- Repeat from second condition until some condition is met.

3.3 Problem Statement

The test suite reduction problem can be stated as follows:

A. **Given**. A test suite T of test cases $t1, t2, t3, ..., tm$, a set of testing requirements $r1, r2, r3, ..., rn$ that must be satisfied to provide the desired test coverage of the program, and subsets $T1, T2, .., Tn$ of T, one associated with each of the ris such that any one of the tests tj belonging to Ti satisfies ri.

B. **Problem**. Find a minimal cardinality subset of T that exercises all ris exercised by the complete test suite T.

Keywords:Genetic Algorithm,Test Case Reduction,Software Testing,Test Requirement matrix,Representative set.

3.4 Framework

Basic Layout : In this section we have shown the basic layout of our framework.How our algorithm will begin with available test requirement matrix generated by our code for any specific program and then initial population is created.Fitness value of each chromosome is calculated and then the best parents are being selected from the available population on the basis of their fitness value.Now Crossover is applied on the selected parents to generate two new children.Mutation operation is being performed on the parent to obtain variety in the chromosome so that better population could be generated.

Each step being discussed above are mentioned and described in their respective modules.

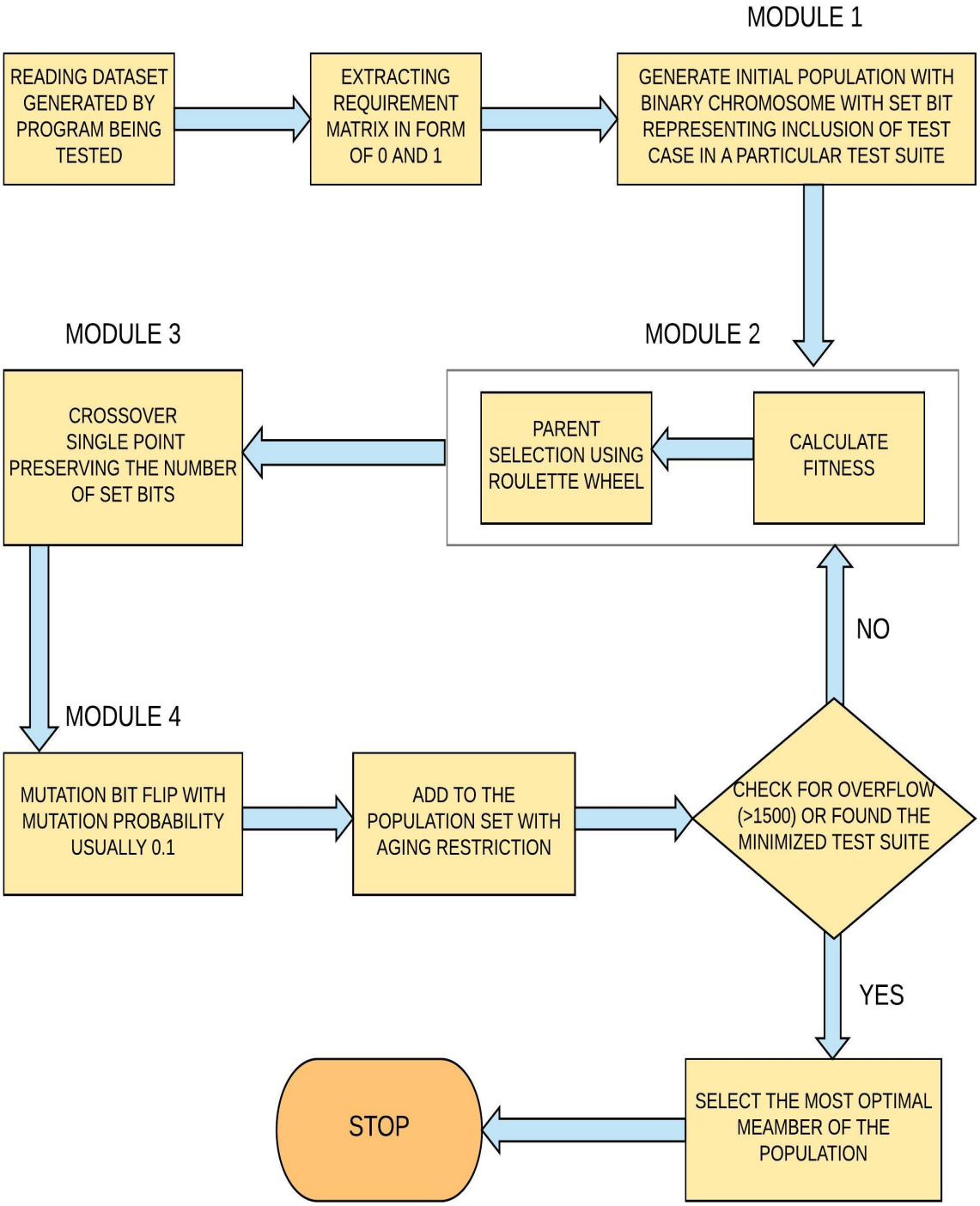

Figure 1: Frame Work

Module1

There are two primary methods to initialize a population in a GA. They are

- **Random Initialization :**Populate the initial population with completely random solutions.
- **Heuristic initialization :** Populate the initial population using a known heuristic for the problem.

It has been observed that the entire population should not be initialized using a heuristic, as it can result in the population having similar solutions and very little diversity. It has been experimentally observed that the random solutions are the ones to drive the population to optimality. Therefore, with heuristic initialization, we just seed the population with a couple of good solutions, filling up the rest with random solutions rather than filling the entire population with heuristic based solutions.

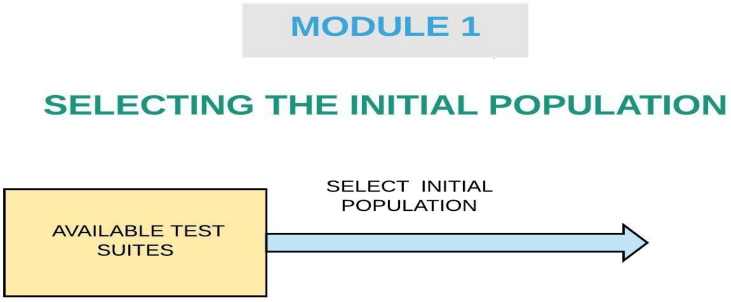

Figure 2: Selection of Initial Population

Module2

The fitness function simply defined is a function which takes a candidate solution to the problem as input and produces as output how fit our how good the solution is with respect to the problem in consideration.

Calculation of fitness value is done repeatedly in a GA and therefore it should be sufficiently fast. A slow computation of the fitness value can adversely affect a GA and make it exceptionally slow.

In most cases the fitness function and the objective function are the same as the objective is to either maximize or minimize the given objective function. However, for more complex problems with multiple objectives and constraints, an Algorithm Designer might choose to have a different fitness function.

A fitness function should possess the following characteristics :

- The fitness function should be sufficiently fast to compute.
- It must quantitatively measure how fit a given solution is or how fit individuals can be produced from the given solution.

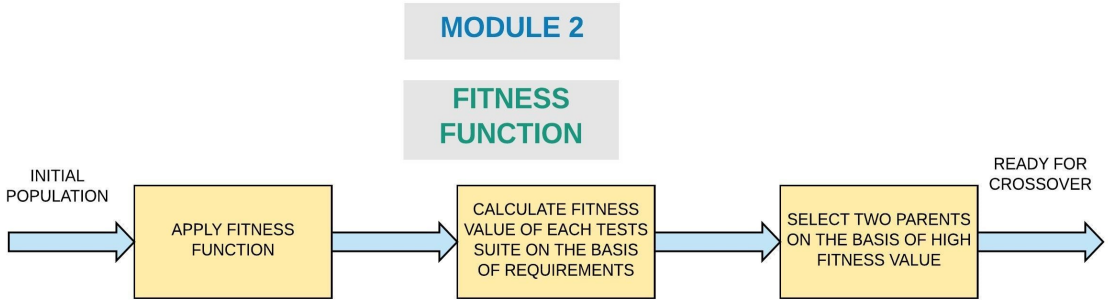

Figure 3: Fitness Function

Module3

In genetic algorithms, crossover is a genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next.The crossover operator is analogous to reproduction and biological crossover. In this more than one parent is selected and one or more off-springs are produced using the genetic material of the parents. It is to be noted that these crossover operators are very generic and we might choose to implement a problem-specific crossover operator as well.

One Point Crossover : In this one-point crossover, a random crossover point is selected and the tails of its two parents are swapped to get new off-springs.If the parents had always the same size and shape, this operation could be performed in a single stage, by selecting any link as the crossover point.

- first the two parent trees are traversed to identify the parts with the same shape, i.e. with the same arity in the nodes encountered traversing the trees from the root node, then
- a random crossover point is selected with a uniform probability among the links belonging to the common parts identified in previous step.

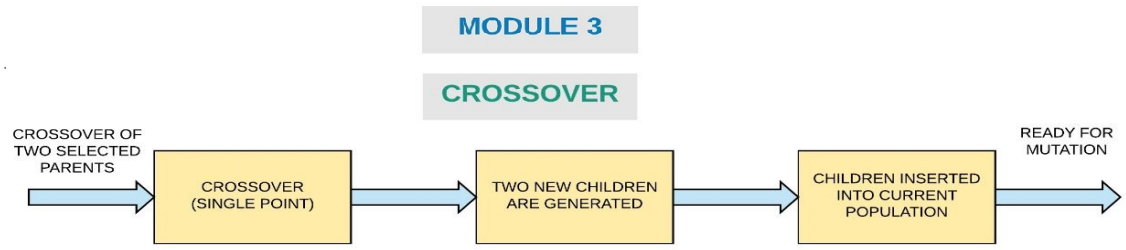

Figure 4: Crossover

Module4

In simple terms, mutation may be defined as a small random tweak in the chromosome, to get a new solution. It is used to maintain and introduce diversity in the genetic population and is usually applied with a low probability.

Mutation is the part of the GA which is related to the exploration of the search space. It has been observed that mutation is essential to the convergence of the Genetic Algorithm while crossover is not.

Mutations stimulate a population that moves toward the goal in leaps and bounds, other times, the mutation slow road in wrong direction.With each succeeding generation, populations are progressively improved by crossover, copying, mutation and extinction. The exploitation of small differences in fitness yields major improvements over many generations in much the same way that a small interest rate yields large growth when compounded over decades.

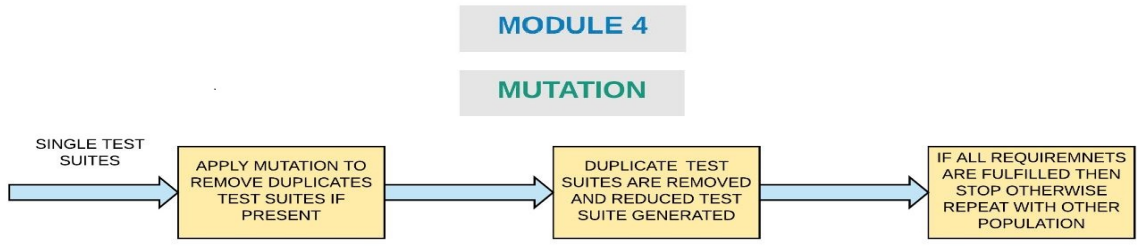

Figure 5: Mutation

Chapter 4

Proposed Approach

Before creation of initial population, the algorithm needs a test requirement matrix. Test requirement matrix (TR) is a two dimensional 0-1 matrix of size ($m \times n$). The test suite T= t1, t2, t3 .....,tm is represented in row and the requirement R=r1, r2,.....,rn is represented in the column. That is each row of the matrix represent requirements fulfill by a particular test case. Entry into the TR matrix is determined by TR(i,j).

Test case	Requirements to be satisfied					
No	r1	r2	r3	r4	r5	r6
t1	1	1	1	0	0	0
t2	0	1	1	1	1	0
t3	1	0	0	0	0	1
t4	0	0	1	0	0	1

Figure 6: Test Requirement Matrix

As for the $0-1$ matrix with m rows and n columns, it is essential to select a subset of rows to cover all of the columns in the matrix with minimal cost. Suppose the vector element represents the row i in the vector x is selected and xi=0 means not, therefore, the set coverage problem can be represented as standard optimization problem: $Minz(x)=$ s.t $i = 1, 2, 3, 4, ...$ (Ensure that every column is covered by at least one row) x_j s.t. $j = 1, 2, 3, ...$ The test suite reduction problem is converted to set coverage problem, and then converted to standard optimization problem. The idea of proposed algorithm start from this. It is an optimization algorithm that can use genetic algorithm to solve this reduction problem. The GA process is represented in the flow chart given in figure 2.

Require: TR:Test Requirement Matrix $n \times m$
Ensure: All test cases together cover all requirements

for $i = 1$ **to** n **do**
$S \leftarrow GA(i)$
if $feasible(S)$ **then**
  break
else
  continue
end if
end

Algorithm 1: Test Case Reduction

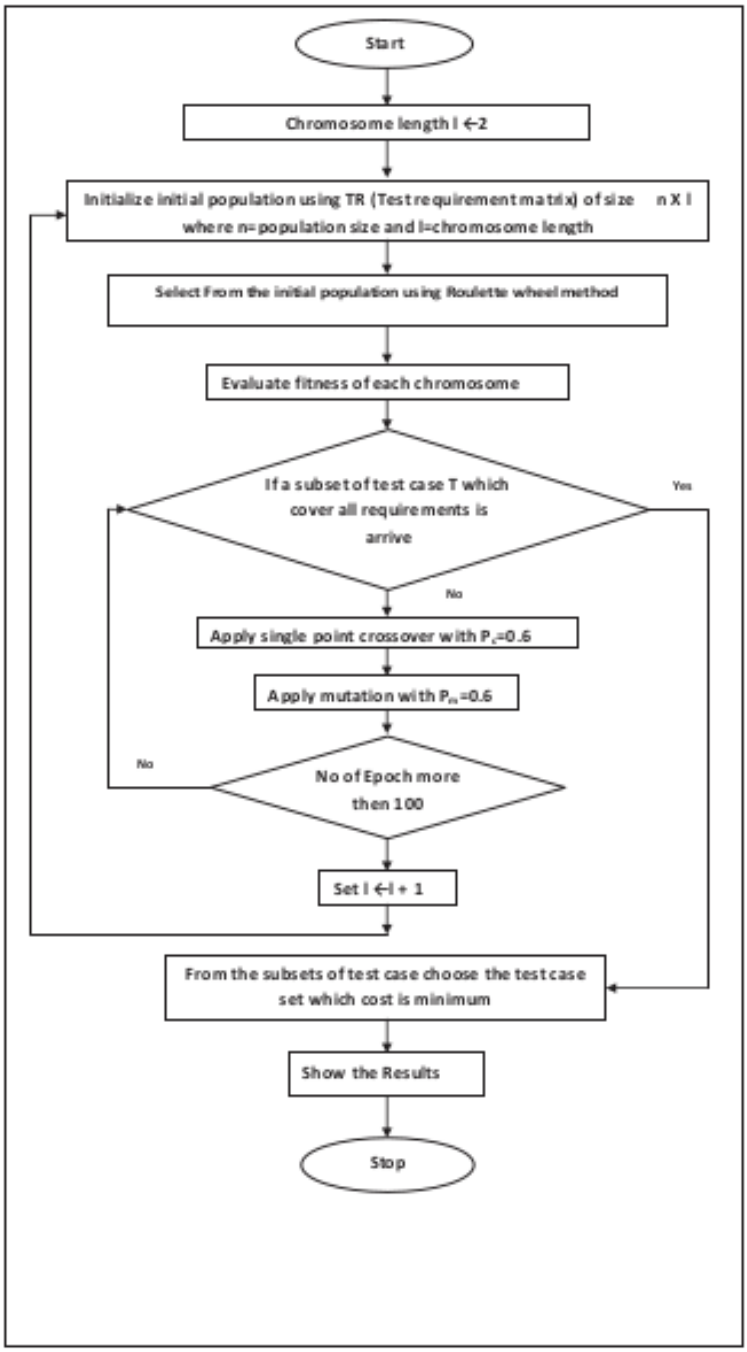

Figure 7: Proposed Algorithm

Require: Integer x representing the number of testcases seeked
Ensure: $y = x^n$
$S \leftarrow initialpopulation()$
while $S.size() \leq 2500$ **do**
$x \leftarrow$ Parent selection using roulette wheel
$y \leftarrow$ Parent selection using roulette wheel
$x\prime, y\prime \leftarrow crossover(x, y)$
$mutate(x\prime)$
$mutate(y\prime)$
$calcfitness(x\prime)$
$calcfitness(y\prime)$
$S.insert(x\prime)$
$S.insert(y\prime)$
end while
$R = max(fitness(S))$
return S

Algorithm 2: Genetic algorithm part

C.Initial Population: Each chromosome of the initial population represents a set of test case i.e a test suite. The initial population is built up randomly using the test case pool. We use permutation encoding for encoding the chromosomes. Each chromosome contains a set of test case as given in fig 3.

The entire population should not be initialized using a heuristic, as it can result in the population having similar solutions and very little diversity. It has been experimentally observed that the random solutions are the ones to drive the population to optimality. Therefore, with heuristic initialization, we just seed the population with a couple of good solutions, filling up the rest with random solutions rather than filling the entire population with heuristic based solutions.

D. Selection

Parent Selection is the process of selecting parents which mate and recombine to create

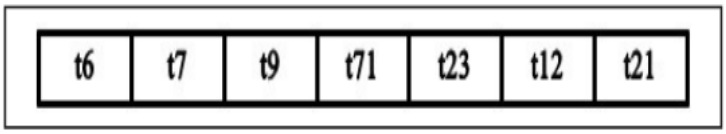

Figure 8: Initial Population

off-springs for the next generation. Parent selection is very crucial to the convergence rate of the GA as good parents drive individuals to a better and fitter solutions.

We use rank selection to select the chromosome to go to the next epoch. Elitism is used as test show that best population are selected.

However, care should be taken to prevent one extremely fit solution from taking over the entire population in a few generations, as this leads to the solutions being close to one another in the solution space thereby leading to a loss of diversity. Maintaining good diversity in the population is extremely crucial for the success of a GA. This taking up of the entire population by one extremely fit solution is known as premature convergence and is an undesirable condition in a GA.

E.Crossover The crossover operator is analogous to reproduction and biological crossover. In this more than one parent is selected and one or more off-springs are produced using the genetic material of the parents. Crossover is usually applied in a GA with a high probability .

Since the chromosomes are binary and have fixed number of set bits in a single iteration, crossover method varies from traditional methods. For we will take crossover probability $\times$ number of set bits $\rightarrow \alpha$ number of set bits from left in both chromosomes, and then flip the rest of the set bits. This assures that the chromosomes also have the same number of set bits.

F. Mutation Mutation is defined as a small random change in the chromosome, to get a new solution. It is used to maintain and introduce diversity in the genetic population

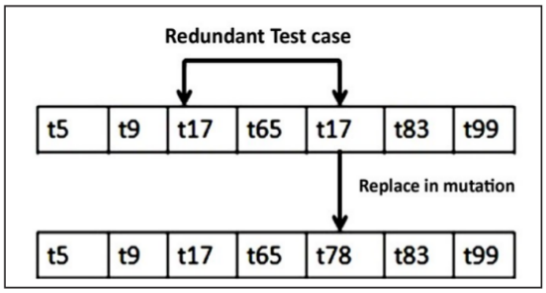

Figure 9: Mutation Operation

and is usually applied with a low probability. If the probability is very high, the GA gets reduced to a random search.

Mutation is used to replace the duplicate test case present in the test suite. For duplicate test case the algorithm randomly select a test case from the existing set that are not included in the chromosome with a mutation a probability of 0.2 .

G. Fitness value The fitness value of each chromosome is calculated by performing and operation among all the requirement sets of individual test case. Then fitness the result is converted into a percentage which denotes how much percentage of requirements is covered by the chromosome. This percentage is calculated using the given below equation.

$$F(x) = \frac{\text{Number of Requirements Fulfilled}}{\text{Total number of Requirements}} \times 100$$

Chapter 5

Experimental Setup And Results

5.1 Dataset Used

We have obtained over data set from [3] which includes the no. of test suites used and the requirements that are to be fulfilled and final reduced data set size. We obtained different datas for various programs like AVL tree,Red Black tree and also for basic triangle validation program.

5.2 Tools and System Used

All experiment will be run on a machine with 2.4 GHz with 8GB of RAM.We have implemented over algorithm in C++ and with inbuilt functions of GA on MATLAB.

The available inbuilt functions of MATLAB are being analysed and compare with that of our implementation through different proposed algorithm and variations.

5.3 Work done and Results obtained

We have implemented the code for all different sections of Genetic Algorithm like Initial Population,Parent Selection,Crossover and Mutation.We have used a variant of Genetic Algorithm.

The results obtained contains the size of representative set (Reduced Test Suite) and the desired result is represented in the form of binary string.The time period required for performing this task is also recorded for various size of initial population and compared with available data sets.

Also we have plotted bar graphs for various programs and have seen the trade-off between the test suite size and time taken for the algorithm to find the representative suite.Another comparison is shown for these programs for different initial population.

The benchmarking has be done on AVL tree program to show that how our proposed approach derives better results than those which are already available in terms time taken for different test suite size and at various initial population.

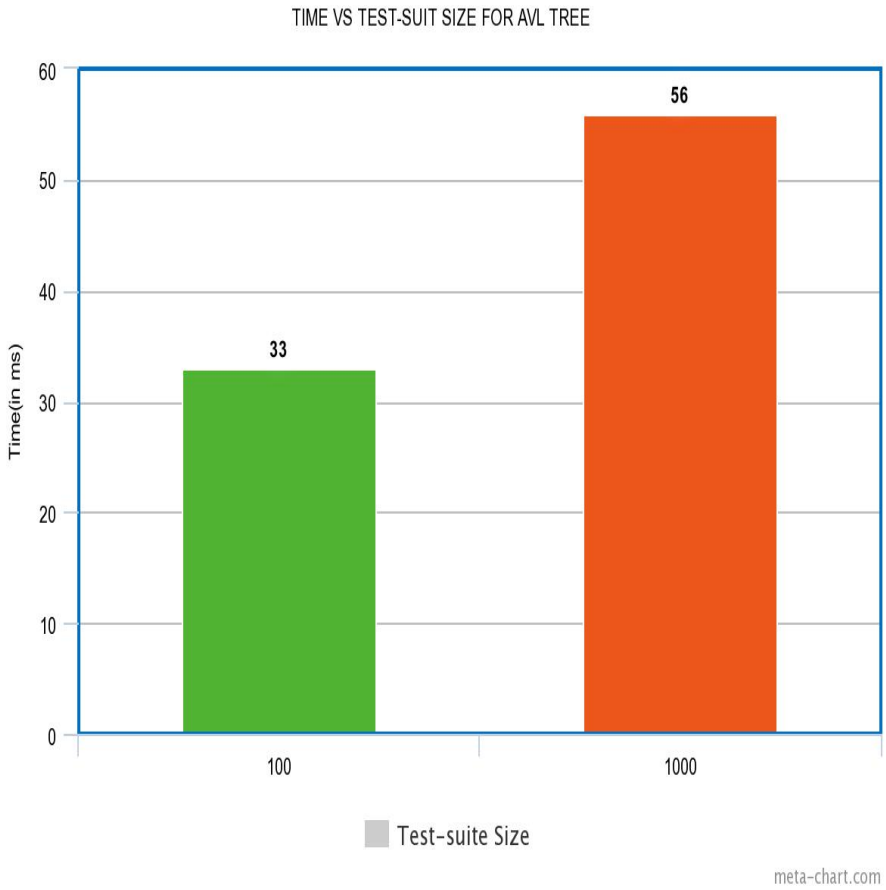

Figure 10: AVL

Table 1: Results for AVL Tree

Test Suite Size	**Time(ms)**
100	33
1000	56

Table 2: Results for RedBlack Tree

Test Suite Size	**Time(ms)**
100	35
1000	61

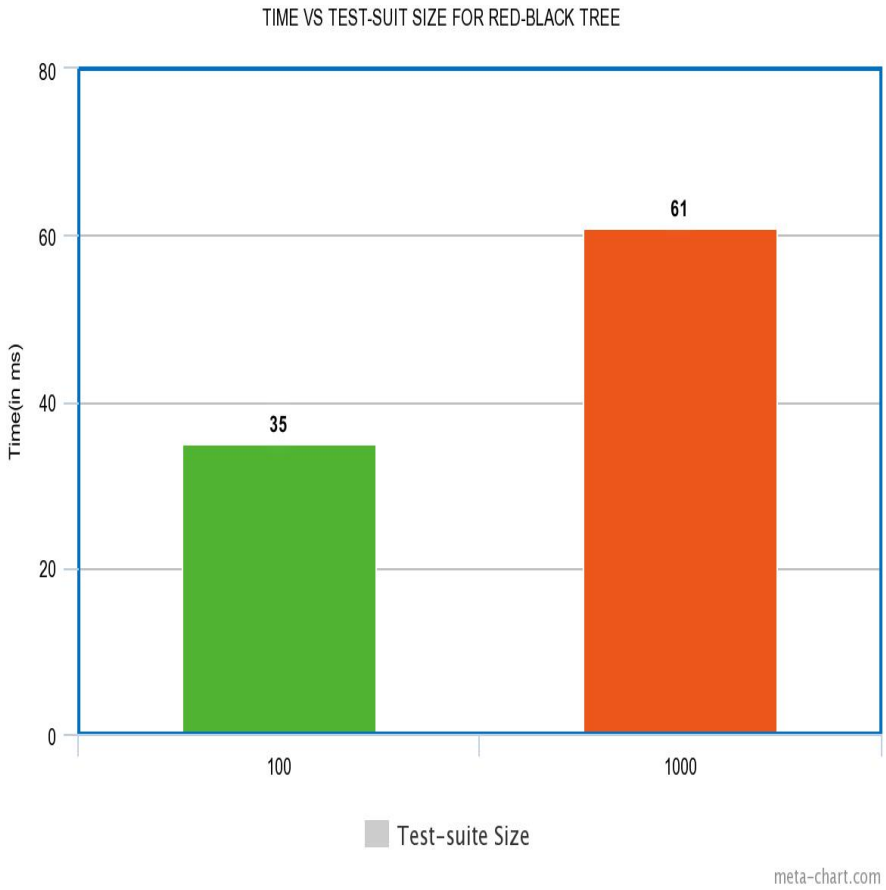

Figure 11: RedBlack

Table 3: Results for Triangle problem

Test Suite Size	**Time(ms)**
100	62
1000	88

Table 4: Comparison of results

Program	**Time(ms)**
Our proposed solution	33.08
Benchmarking through available results	50

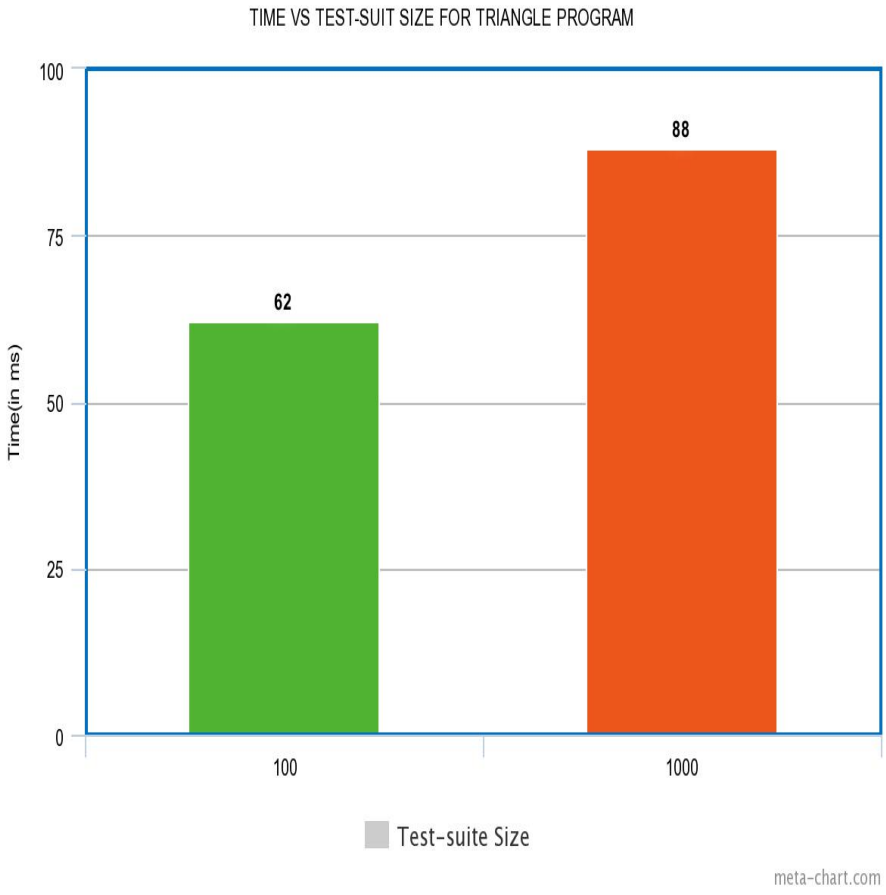

Figure 12: triangle

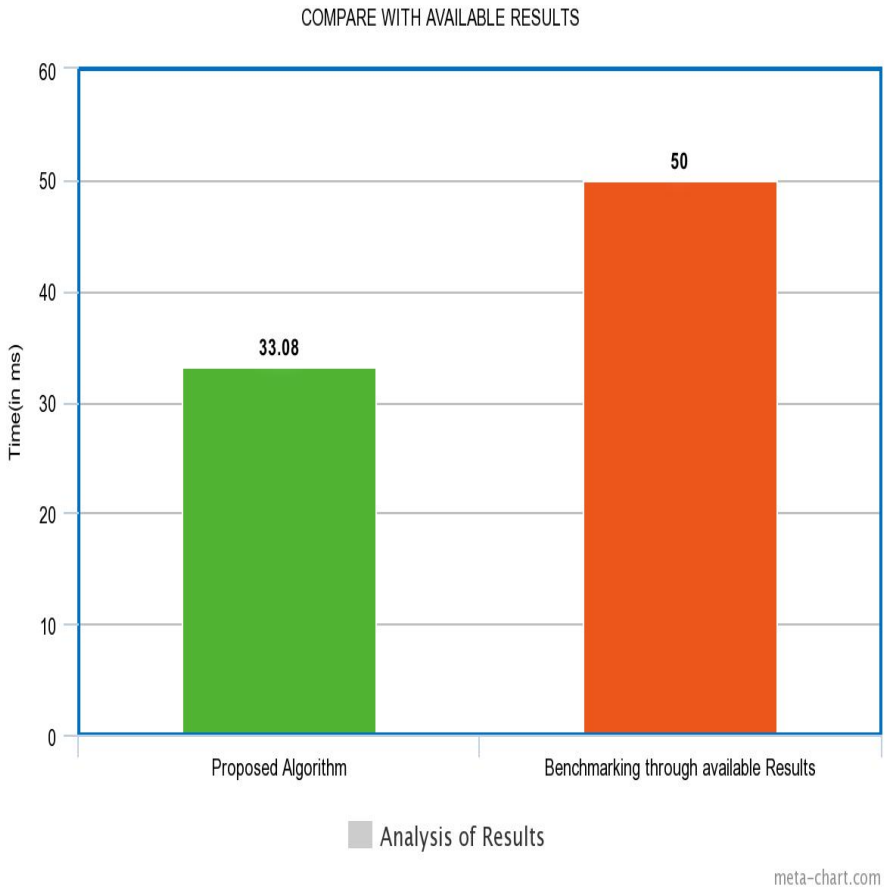

Figure 13: Benchmarking

vs test suite size150 initial population.jpeg vs test suite size150 initial population.jpeg

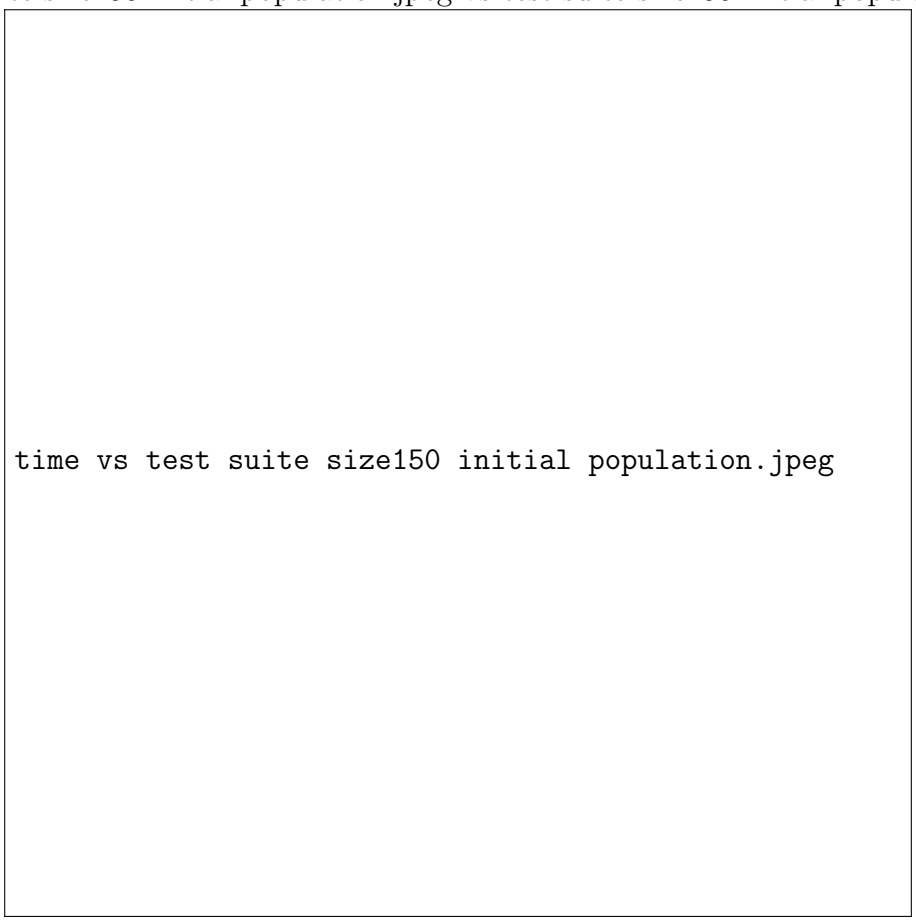

Figure 14: Time vs test suit size with initial population size as 150

set size.jpeg set size.jpeg

Figure 15: Representative Set Size

Chapter 6

Conclusions

In this report we have used a variant of Genetic Algorithm to reduce test suite size from a give test suite which fulfills all the requirements of a program.This is an optimization problem.We have optimized some problems and have compared over results to the standard available results.

The proposed work can be modified using different variants and operators and comparing the results obtained to that with the standards.This technique could be used for various purpose like path coverage in programs.

Chapter 7

Future Work

Although our initial studies are encouraging, much more experimentation must be conducted to verify the effectiveness of our techniques in general and in practice. Another experiments we should do to further investigate the fault-detection capabilities ties of a block-based test adequate test-suite on software. These studies will also let us evaluate our algorithms and help us provide guidelines for test-suite reduction in practice.Genetic Algorithm can be used for multi-modal optimization in which we have to find multiple optimum solutions.Here we have calculate our test requirement matrix with the help of counter on each requirement,in future we can make a tool that with automatically generate test requirement matrix.

References

[1] Bin-Kui Sheng, Z.-f. H. A genetic algorithm for test-suite reduction. Master's thesis, Dept. of Information Management, Zhejiang University of Finance Economics Hangzhou, P. R. China., 2014.

[2] E.Goldberg, D. Genetic algorithm in search,optimization and machine learning. In *Congress on Evolutionary Computation, IEEE* (1989), pp. 2516–2568.

[3] http://hackerrank.com. *Data Structure in Competitive Programming.*

[4] Mitchell, M. *An Introduction to Genetic Algorithms.* 1995, ch. Genetic algorithm in Problem Solving, pp. 1–62.

[5] Shaout, M. A. D. S. A. Test case reduction techniques-surveys. Master's thesis, The University of Michigan - Dearborn Michigan - Dearborn, 2016.

[6] Xue-ying Ma, B.-k. S., and qing Ye, C. Test-suite reduction using genetic algorithm. Master's thesis, College of Computer Science and Technology, Zhejiang University, Hangzhou ,China, 2012.