8-2017

Boundary Value Analysis for Input Variables with Functional Dependency

Manmohan Maheshwari
Harrisburg University of Science and Technology

Recommended Citation

Maheshwari, M. (2017). *Boundary Value Analysis for Input Variables with Functional Dependency*. Retrieved from http://digitalcommons.harrisburgu.edu/csms_dandt/2

Boundary Value Analysis for Input Variables with Functional Dependency

Submitted to Harrisburg University for Science and Technology as a Partial fulfillment of the Requirements for the Master of Science degree in Computer Science

BY:

Manmohan Maheshwari

manukhator@gmail.com

Supervised by

Nushwan Al-Nakash, PhD.

Assistant Professor of Computer Science

Nal-nakash@harrisburgu.edu

August 2017

Table of Contents

ABSTRACT

Software in today's world is used more and in different ways as well than ever before. From microwaves and vehicles to space rockets and smart cards. Usually, a software programmer goes through a certain process to establish a software that will follow a given specification. Despite the hard work of the programmer, sometimes they make mistakes or sometimes they forget to include all the possibilities of the question for which they are writing the program, which is very humanly in nature. And for those mistakes, a testing unit is always there.

There are numerous techniques of Software Testing, one of which is Boundary Value Analysis. A modified version of Boundary Value Analysis using input parameters with functional dependency is proposed in this work. The idea is derived from the inter dependency of functions among the input parameters. With this modified algorithm, an automated testing tool is created and implemented. This testing tool shows the advantages of the modified algorithm developed over the Functional Tree Approach and reduces a significant amount of test cases that leads to an exhaustive testing. This modified method will test almost every possible required test case increasing the system's efficiency. This method will be a very good help for any product based company saving a huge amount of money and time.

Generalized BVA generates 5*n number of test cases where n is number of variables while Function Tree method generates the highest of all three that is n*5^(n-1) and the modified approach generates 7*n + k number of test cases where k is the number of mutants killed at each step. So, it shows that the number of test cases in case of modified algorithm is significantly lower than the Function Tree algorithm while almost similar as regular BVA but it covers more functionalities and features

***Keywords*:** Software Testing, Boundary Value Analysis, Functional Dependency, Functional Tree, Mutation Testing, Automation Testing.

LIST OF FIGURES

CHAPTER – 1

1.1 Introduction

Despite the hard work of the programmer, sometimes they make mistakes or sometimes they forget to include all the possibilities of the question for which they are writing the program, which is very humanly in nature. And for those mistakes, a testing unit is always there. The job of this unit is to figure out what mistake has the programmer made or what did he/she forget to include. In some of the big organizations, they have huge teams of testing for their products because they have an important consideration in mind of the consequences of a software error.

Most of the software usually need to stick to a single rule, i.e. to make sure that what is expected, it does. To use all the available resources in better sense, computers should also be helpful in "the art of software testing" to an improved extent, than is currently the case today. One of the issues today is if humans can make errors in coding then they can also make errors in software testing.

The solution of this is not to remove human beings from the process of software testing but to use today's software development art form and make computers also participate. This thesis will present research aimed at classifying, examining, and improving the basic concept of boundary value analysis through automated software testing. To describe it further, we should first talk about software testing, boundary value analysis, functional dependencies and then possibility of creating a method that will solve the problem.

Software testing is defined as *a formal process in which a software unit, several integrated software units or an entire package are examined by running the programs on a computer. All the associated tests are performed according to approved test procedures on approved test cases (Galin, 2004).* Software testing, is to test or check whether the software is executing perfectly and the necessary requirements are fulfilled. It is a human tendency to make errors and for elimination of these, tests are done on the product being developed to find out the problem in the software, that is why software testing is necessary.

Software Testing is very expensive method in terms of time and money but it is the only way to find out bugs in the system. Due to time and budget constraints, it is impossible for us to perform exhausting testing for every set of test data, especially when there are enormous pools of input combinations. It requires an easy way or special techniques which will select test cases intelligently from the pool of cases, so that every test scenario is covered. So, optimal testing is necessary to save time and money.

The hard part in this procedure of testing is to generate the test cases. A test case is a condition put on each input parameter and those sets of conditions will give the tester an output which will help in the testing objective. A good testing technique is the one which covers every aspect of the requirement and the objective of testing. In the Late 90's test cases were derived manually but for some products this procedure takes time much more than the time of required for the development. So, the method of automated testing is introduced to test the product automatically using program. Detecting and removing errors in the earlier phases will even reduce the cost of whole development.

1.2 Boundary Value Analysis

There are numerous techniques of Software Testing, one of which is Boundary Value Analysis. Regarding boundary value analysis, NIST defines it as [6], a selection technique in which test data are chosen to lie along 'boundaries' of the input domain [or output range] classes, data structures, procedure parameters. The basic idea of boundary value analysis can be judged from the word boundary, as we know most of things in this world have boundaries. Such bounded values are used in this procedure. This process tests the product being developed on the boundary values of each input parameter and then generate the desired test cases.

The basic idea of boundary value analysis(BVA) is to generate the number of test cases using the parameter values at their boundary points such as minimum (min) and maximum (max), the values next to the boundary points such as just above the minimum (min +) and just below the maximum (max −) and the median i.e. the nominal value (nom).

In a program, the input parameters consist of the upper and lower bounds, so the test cases are obtained following the boundary value analysis, by holding the values of all but one parameter at their nominal values and letting that parameter assume its extreme values [1].

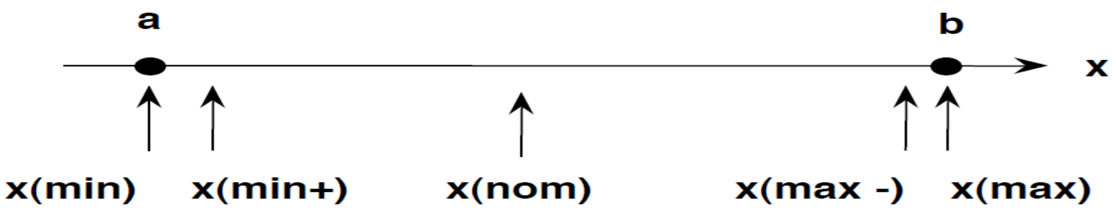


Fig 1.1 [4]: Boundary Value Analysis Range

1.3 Functional Dependencies

There are various examples of programs on which boundary value analysis works, one of which is Next Date function. Though it seems like a perfect method for software testing but it has some limitations too. It does not work on input parameters with functional dependencies e.g. X, Y and Z are three input parameters, now Y = f(X) and Z = g (X, Y), thus, Y is a function of X and Z is a function of X and Y, so the above method may not be applied. These input parameters with functional dependencies were studied previously. One of the previous methods in [2] uses the boundary value analysis with divide and rule approach. The other method in [3] uses the function tree for generalization of boundary value analysis to input parameters that are functionally dependent.

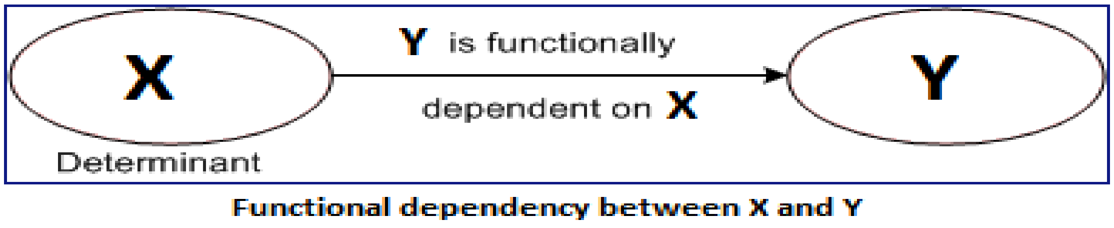


Fig 1.2 [5]: Functional Dependencies

1.4 Automation Testing

The aspect of automation in software testing is focused on keeping human involvement to a minimum. Software Test automation uses specialized tools for the execution of tests of the product and compares the actual results against the expected result. But the question here is, why we need automation testing. Well, automation testing is important because of many reasons. One such reason is that it increases the speed of the testing process by executing test cases on its own. As we know, manual testing is both time and cost consuming and can be boring sometimes. An example for difficulty in manually test is testing for multi lingual sites. For that, automation testing is required. One of the advantages of automation is that we can run the testing overnight even unattended as it does not require human interaction.

So again, through this thesis I am trying to make the technique of boundary value analysis more efficient and easy while making the process automated. It is a very difficult job to make an automation tool as every piece of program or every product requires a different automation tool so that it can create only the test cases required for the product. This to go in the aims of the thesis.

1.5 Mutation Testing

The method of Mutation testing is out there for ages and testers have known this for many years. However, few of them are using it for different-different reasons. Most of the procedural steps used are automated, for example, mutant software creation and the white box testing. In this kind of testing, modifying a program code, re-running a suit of correct tests, and deliberately altering are included against the mutated program. The tester can easily assess the quality of a test suite using Mutation testing. So, by mutating different elements in the software in the product's source code and after that checking if the test code can find the mutated errors and detecting those errors.

Mutation testing is a misnomer. Mutation testing tool is a super powerful tool which can be used for checking coverage or detecting testing inadequacies on testing software. It can choose correct mutant programs and going through a comprehensive testing of these correctly chosen programs. It is more of an analytical method than testing technique. For reliable and better results, one should find better methods that can reduce the number of tests and find efficient number of mutants. More efficient numbers of mutants lead to longer testing time duration.

1.6 Problem Statement and Justification

To develop the problem statement, one should know what is the question you are trying to answer and why. In this work, I am trying to question whether the testing techniques nowadays are enough to answer or test all the possible outcomes that a program has. Can I make an automation tool that will help me solve this problem of efficient testing? With the functional dependencies of the attributes, is it possible to create a testing technique that will be far much better than normal Boundary Value Analysis?

Now after defining the problem, we should discuss why this problem should be solved and who all are affected with this problem solving is discussed. In this thesis, the problem is, can I make an automation tool that will help me solve this problem of efficient testing? And with the functional dependencies of the attributes, is it possible to create a testing technique that will be far much better than normal Boundary Value Analysis?

It is very important that testers of any field should be able to write test cases based on Boundary value analysis because the hardest part in the procedure of testing is to generate enough test cases which can test all the possibilities of the question for which they are writing the program. Testing components will always be a challenging area for research. During the design, the component properties are going to be adopted. And one of the motive is to validate and test against these adopted component properties. We can gain higher productivity with generic components. To test a generic component which contains several generic parameters is very hard to test. Therefore, all possible applications through customization should be tested.

There are various problems regarding testing in this world. For boundary value analysis, an efficient number of test cases should be generated which will check for all the wrong and right possible outcomes. But with inputs with functional dependencies, sometimes test cases fail to check all the outcomes efficiently. So, in this thesis, a modified approach is created which will check for test cases for inputs with dependencies. This modified approach will be created using two already present techniques known as “Divide-and-Conquer-Approach” and “The Function

Tree”. Then an automation tool will be created which will first use mutation testing to check and then automatically create all the test cases for those input variables.

Boundary value analysis is another type of Black box testing and is also a part of stress and negative testing. By the name, boundary value analysis indicates limitation on something. So, when the inputs are supplied within the boundary values, it is positive testing and inputs are considered as negative testing beyond the boundary values. For example, if an application accepts VLAN Ids ranging from 0–255, then 0, 255 will be the boundary values and any input going below 0 or above 255 will be invalid and will constitute negative testing.

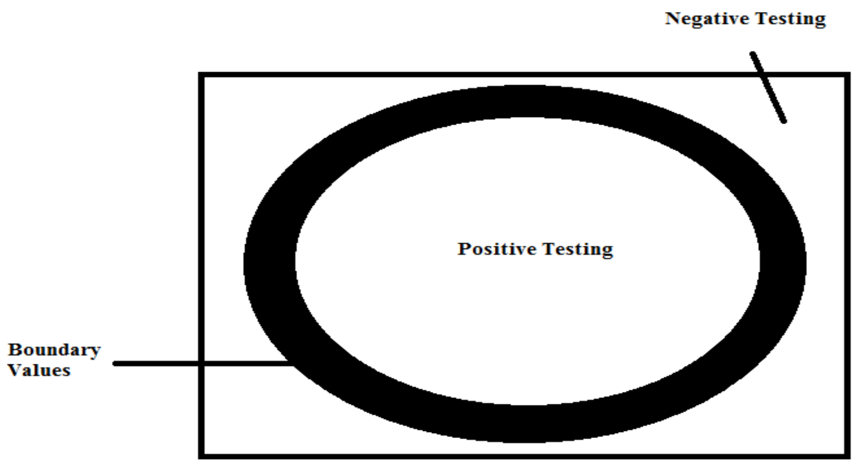


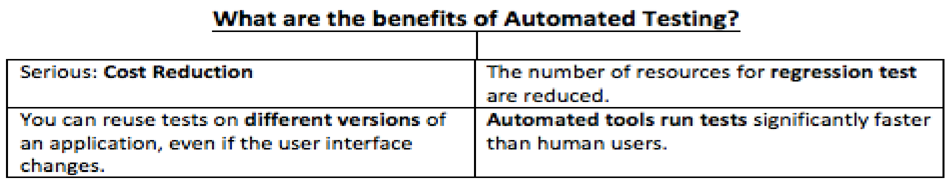
Fig 1.3: Defining the boundary of the values

This technique of Boundary Value Analysis is used for figuring out the bugs that usually occurs at the boundaries rather than ones that exist in the center. This technique is used to reduce the number of test cases to minimal, while assuring that the test cases selected are effective test cases which would help in testing whole scenarios.

To continuously maintain and improve the efficiency and quality of the software system’s development is a very hard challenge for any company or organization. In many software projects, because of time or cost constraints organizations neglect the testing phase. Sometimes, this might lead to customer dissatisfaction, followed by a lack of product quality and ultimately to increased overall quality costs.

Despite the hard work of the programmer and tester, an automation tool will reduce the man power in some extent and help the industry save money. Poor test strategy, delay in testing, underestimated effort of test case generation and subsequent test maintenance might lead to additional cost of the software project.

Automated tests are fast and can run frequently within different tests due to reused modules, and for the software products with a long maintenance life, it is cost-effective. While testing in an agile environment, changing software requirements of the system is necessary. As the software project progresses, test cases should be modified for a period in both manual and automated testing. It is important that people should know that using test automation, complete coverage of all the tests is unrealistic. Test automation allows performing different types of testing efficiently and effectively.


What are the benefits of Automated Testing?

Serious: **Cost Reduction**	The number of resources for **regression test** are reduced.
You can reuse tests on **different versions** of an application, even if the user interface changes.	**Automated tools run tests** significantly faster than human users.

Fig 1.4 [9]: Benefits of Automated Testing

The main reason of using the automation testing is that “Automated regression tests which ensure the continuous system stability and functionality after changes to the software were made lead to shorter development cycles combined with better quality software and thus the benefits of automated testing quickly outgain the initial costs” [8].

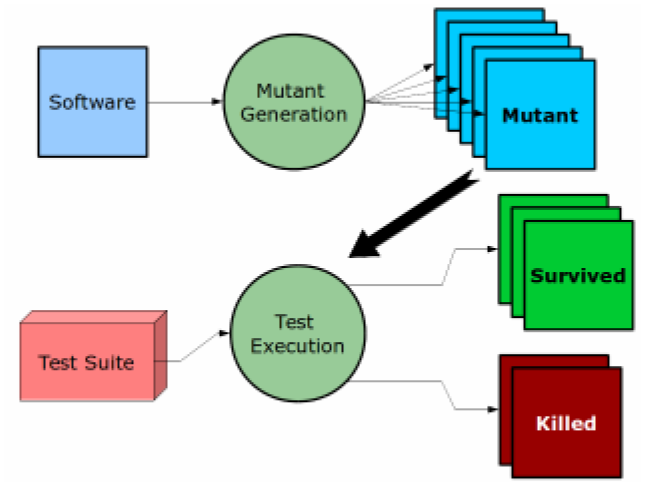


Fig 1.5 [7]: Mutation Testing and its Test Suits

Manual testing can be mundane and exasperating while test automation can remove all the frustrations of a tester and allows test execution without human interaction guaranteeing accuracy and testers can now concentrate on more difficult test scenarios.

Sometimes it happens that your tests can not find a bug, will you believe that there are no bugs? For that, one of the tests that allow you to assess the tests is Mutation testing. Unlike the other testing methods which will give you a proper testing result, in mutation testing there is no Pass or Fail result of disposition even if the output fails in the test cases to meet the standard output. Instead, to improve the lack of whatever is causing it, adjustments are made, then the test is done again until we attain a satisfactory score.

CHAPTER – 2

In the previous chapter, a description to the effect of solving the problem and who all will the effort affect. There have already been some ground-breaking discoveries in this matter such as techniques like "Divide-and-Conquer-Approach" which deals with inter-dependent parameters and "The Function Tree" which uses a technique in which there are various levels of testing the functional dependencies.

2.1 DIVIDE AND RULE APPROACH [6]

This algorithm named divide and rule creates various independent sets of parameters by breaking the dependencies among those parameters. This method will make sure that there is no interference within the boundary values or with the boundary values of each other by the parameters in the new independent sets. And this is the reason, why this technique is named Divide and-Rule approach [2]. Using this method, different independent parameter sets can be created from the dependent parameter sets and after that on these independent parameter sets, the traditional method of boundary value analysis is performed.

This technique usually deals with the inter-dependent parameters which means that the boundary values of those parameters are dependent on the value of some other parameter. So, such type of dependency in parameters is known as boundary dependency. In this approach, there are 3 types of parameters: Dependent Parameters, Independent Parameters, and Boundary-determining Parameters. In the next section, written is an algorithm which helps in creating the test cases on boundary value analysis using divide-and rule approach.

2.1.1 Algorithm

This algorithm is based on the idea that a multiple set of independent parameters can be generated from a single set of dependent parameters. Using the simple or traditional boundary value analysis will not generate various test cases that are required for testing all possibilities. Therefore, a generic algorithm is required so that we will be able to generate the independent sets of parameters with any kind of dependency. This method can be used for generating test cases for programs such as Next-Date function [2,12].

This program is based on 6 different types of modules and the program results in six screens with each representing a module of the tool and the six modules are: getVariables, getRelations, getNumOfSets, getBounds, getSetRelations, and generateBVATestCases. So, these six modules are shown in the algorithm as well as in the flow chart representation of this technique as shown in Fig 2.1. User will enter some input and that input will be accepted by each module and then each module will forward its output to the next module. There exists a button for each and every module that is the “Submit Query” button, which will allow the user submission of the input. In the following, the design for each module is presented individually.

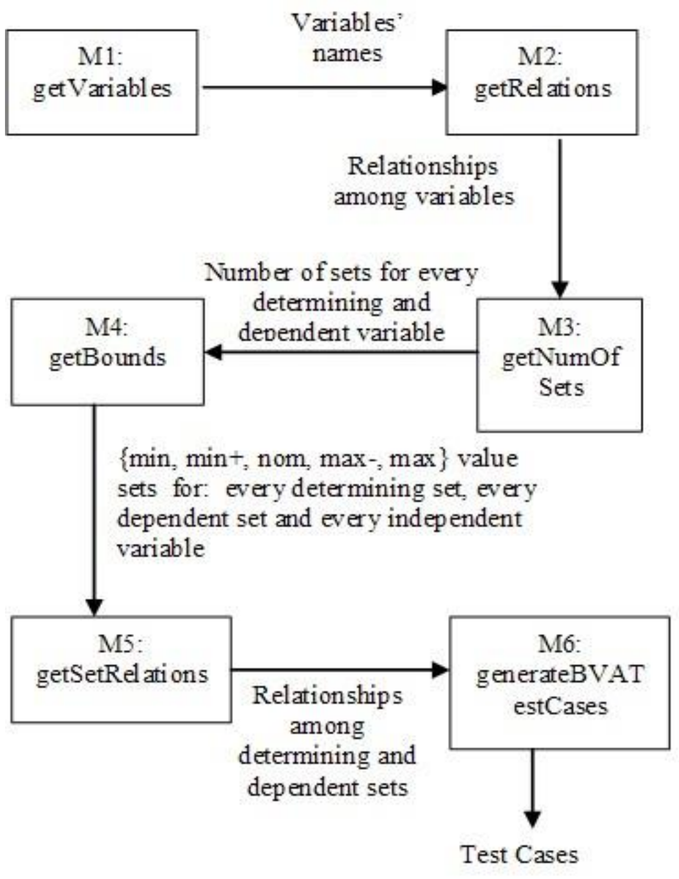

M1: getVariables
Variables' names
M2: getRelations
Relationships among variables
M3: getNumOfSets
Number of sets for every determining and dependent variable
M4: getBounds
{min, min+, nom, max-, max} value sets for: every determining set, every dependent set and every independent variable
M5: getSetRelations
Relationships among determining and dependent sets
M6: generateBVATestCases
Test Cases

Fig 2.1 [2]: System Flow Chart of Divide and Rule Approach

The above representation is explained as [2]:

Module1 (getVariables): In this module, the user is prompted to enter all the variable names. 10 text boxes are displayed in the screen of this module and all the users can enter the names up to 10 variables. Due to some design purposes, there is a limitation on the number of variables.

Module2 (getRelations): In this module, relationships among the variables are defined by the user. Based on the relationships entered by the user, the tool determines the boundary-determining, boundary-dependent, and independent variables. For n variables, the screen displays n^2-1 checkboxes in n rows and n columns. To specify the relationship among variables, the proper checkboxes are checked.

For example, to specify that variable i depends on variable j, the checkbox in ith row and jth column is checked.

Module3 (getNumOfSets): Values for all boundary dependent and boundary-determining variables can be divided into two or more sets. This module prompts the user to enter how many sets can be formed for each boundary-dependent and boundary-determining variable. These sets are referred to as boundary-dependent sets and boundary-determining sets.

For k boundary-dependent variables and j boundary determining variables, the screen displays $k + j$ textboxes.

Module4 (getBounds): This module prompts the user to specify the {min, min+, nom, max−, max} values for every dependent variable, all boundary-dependent sets and all boundary-determining sets.

For p independent variables, $5p$ text boxes are displayed.

For q boundary-dependent variables with $s_1,...,s_q$ sets each, $t_q = 5(s_1 + \cdots + s_q)$ text boxes are displayed.

For r boundary-dependent variables with $s_1,...,s_r$ sets each, $t_r = 5(s_1 \times \cdots \times s_r)$ text boxes are displayed.

Module5 (getSetRelations): This module generates all possible combinations of boundary-determining sets for each boundary-dependent variable. It then prompts the user to associate each combination of the boundary determining sets with a boundary-dependent set. The screen displays all possible boundary-determining set combinations for each boundary-dependent variable. For every combination, it displays *n* radio buttons, where *n* is the total number of sets for that boundary dependent variable.

Module6 (generateBVATestCases): This module generates all possible combinations of all the boundary determining sets. It then associates every combination with a boundary-dependent set of every boundary dependent variable, based on the associations specified by the user in Module 5. It then associates all the combinations with all the independent variables. These boundary-determining set combinations associated with boundary-dependent, and independent variables form the BVA sets. The module then generates the test cases by performing boundary value analysis for every BVA set.

The following algorithm [2] is written in 10 steps:

Step 1: Divide the parameters in three sets:

D: dependent parameters,

B: boundary-determining parameters,

I: independent parameters.

Step 2: For every parameter $d \in D$, create a set of its determining parameters. Each element of this set shall also be an element of set B.

Step 3: For every parameter d ∊ D, create separate sets of all possible boundary value ranges for d. Mark these sets d1 to dx. These sets must contain these bounds (min, min+, nom, max−, max) based on its boundary-determining parameters.

Step 4: For every parameter b ∊ B, create sets of values that will affect its dependent parameter on the boundary values. Mark these sets b1 to by. The relation between sets (d1 to dx) and sets (b1 to by) are specified in Step 6.

Step 5: For every parameter d ∊ D, generate all possible combinations of its sets. For e.g., if a and b are 2 elements of this set and a has 2 determining sets (a1 and a2) and b has 3 determining sets (b1, b2 and b3), then possible combinations will be: a1b1, a1b2, a1b3, a2b1, a2b2, a2b3.

Step 6: Assign a boundary-value set to every possible combination of determining sets. The relation between these sets is that if variables a and b assume the values from a combination apbq, then d can assume a value from corresponding dr only.

Step 7: For boundary-determining sets, generate all possible combinations of all boundary-determining parameters.

Step 8: For every combination, there is in the previous step, assign the boundary values to the sets for each of the dependent parameters using Step6.

Step 9: For every combination, there is, assign all the independent parameters which will create an independent set of combinations.

Step 10: For every independent parameter set that is produced, perform traditional boundary value analysis to generate test cases.

2.2 THE FUNCTION TREE

The function tree approach is used to generate test cases for the input parameters with functional dependencies using boundary value analysis. This method is not for any specific ideology or specific problem but this is very generic solution to the black box testing technique which can use 2, 3 or even many more input parameters dependent on one another. Following are the 3 different levels for implementing this technique.

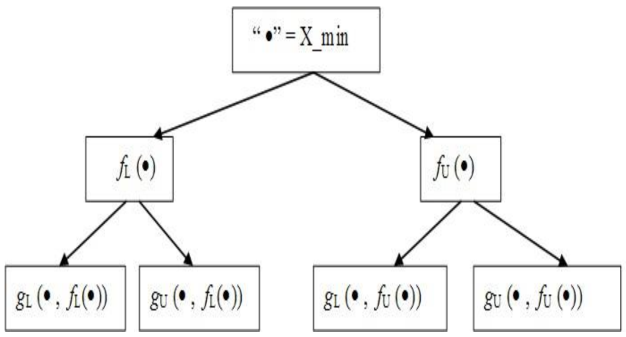


Fig 2.2 [3]: Boundary Function Tree for Xmin

Level 1: Assume that the lower and upper bounds for X are given respectively Xmin and Xmax. Now, the function tree is drawn for each of the two bounds as shown in Fig 2.2. Boundary Test, in this level all the parameters take their boundary values for the implementation and those values include min and max. The function tree for this level is always a binary tree and every path in that tree is a test case. The function tree here is a binary tree which is usually generated by the upper and lower bounds function for *X* and *Y*. Each path on the function tree corresponds to that one test case. There are four testing cases from left to right as:

(Xmin, fL (Xmin), gL (Xmin, fL (Xmin))

(Xmin, fL (Xmin), gU (Xmin, fL (Xmin))

(Xmin, fU (Xmin), gL (Xmin, fU (Xmin))

(Xmin, fU (Xmin), gU (Xmin, fL (Xmin))

There are some test cases that may be same and some may not satisfy the constraint function. The function tree for $X_{\max}$ and function tree for $X_{\min}$ can be drew similarly except that all the $X_{\min}$ is replaced by $X_{\max}$ and that to everywhere in the tree. So, the four testing cases that can be generated from the function tree for $X_{\max}$ are:

(Xmax, fL (Xmax), gL (Xmax, fL (Xmax))

(Xmax, fL (Xmax), gU (Xmax, fL (Xmax))

(Xmax, fU (Xmax), gL (Xmax, fU (Xmax))

(Xmax,fU(Xmax), gU(Xmax,fL(Xmax))

Boundary tests can be formed by remove the duplicated and those that do not satisfy the condition $h\ (X, Y, Z) \geq 0$ or $h\ (X, Y, Z) = 0$.

Level 2: Next-To-Boundary Test, in this level of test at least 1 of the parameters has the next to boundary values that includes max-1 or min+1 and all the other parameters can have any value. So, in total there are 4 values assigned to any parameter which includes min, max, min+1, max-1.

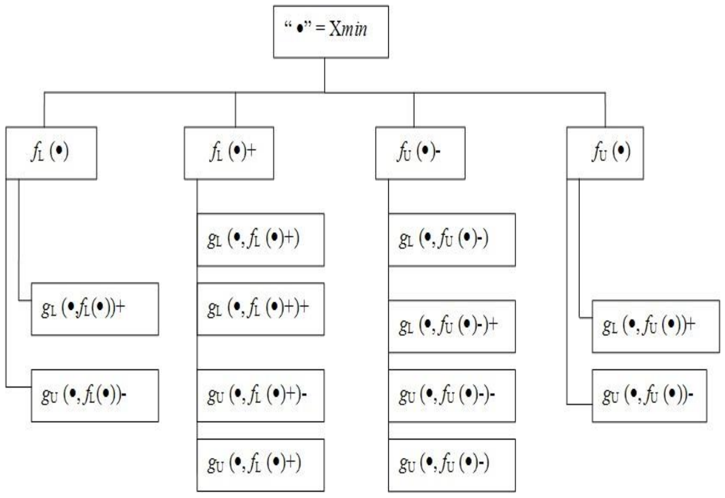


Fig 2.3 [3]: Next-to-boundary Function Tree for Xmin

Next-to-Boundary tests can be formed by selecting the one's that has at least one of parameters is not boundary value and remove the duplicated and those that do not satisfy the condition $h\ (X, Y, Z) \geq 0$ or $h\ (X, Y, Z) = 0$.

The Next-to-boundary Function Tree for $X_{\max}$ can easily be obtained by replacing $X_{\min}$ with $X_{\max}$. Next-to-boundary Function Trees for $X_{\min}+$ and $X_{\max}-$ would be different with Figure 2. For example, to obtain the tree for $X_{\min}+$, we first replacing $X_{\min}$ in Figure 2 by $X_{\min}+$, then we need to connect the paths from $f_L(\bullet)$ to $g_L(\bullet,f_L(\bullet))$ and from $f_L(\bullet)$ to $g_U(\bullet,f_L(\bullet))$ respectively.

Other the right most side, we need to connect the paths from $f_U(\bullet)$ to $g_L(\bullet,f_U(\bullet))$ and from $f_U(\bullet)$ to $g_U(\bullet,f_U(\bullet))$ respectively. This is because these four test cases were not included in the cases generated in Step 1. Similar changes need to be made to get the Next-to-boundary Function Trees for $X_{\max}-$.

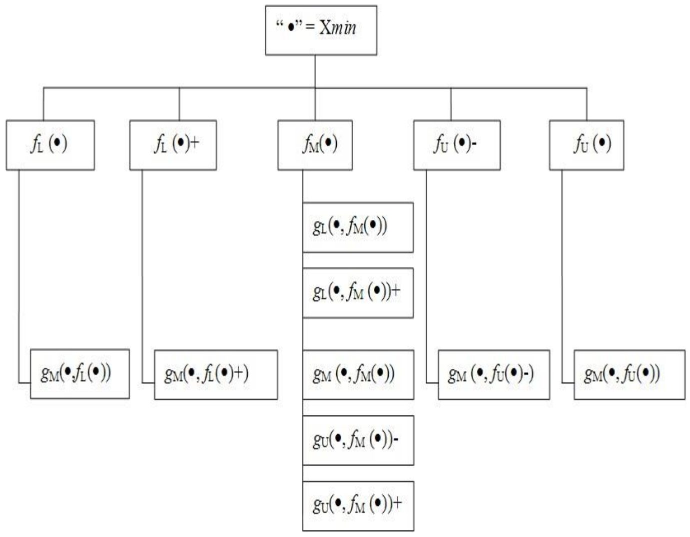


Fig 2.4 [3]: Middle Function Tree for Xmin

Level 3: Normal Test, in this level of test there is a middle value included with the 4 values of Level 2 that is nom. At least one of these parameters has the middle value. This level is almost like traditional boundary value analysis except that in traditional boundary value analysis, all except 1 has the middle value.

Level 3 tests or Middle tests are same as the cases in the traditional boundary value analysis, thus 2 of the 3 parameters taking middle values. The third parameters go through its 5 elements sequence: min, min+, nom, max− and max. Similar as in step 1 and 2, we use the Middle Function Tree to generate test cases in level 3. The middle function tree is the most complete tree. Every element in the *X*'s 5 elements sequence *X*min, *X*min+, *X*nom, *X*max−, and *X*max has a middle function tree.

So, using these 3 levels, an algorithm is designed which will automatically generate test cases. There are two real time systems working on this algorithm.

One of which is Net Provision Activator (NPA), a product of Syndesis Limit where this algorithm is being used for generation of their test cases. And the second one resides in the Atmospheric Physics Lab at Trent University is a Celestial Tracking Software (CTS). It is used to conduct spectral analysis of light from different-different sources such as the Sun, hardware such as spectroscope and mirrors used in the analysis are being controlled by (CTS).

So, shown above are the previously defined work that has been already done but, the Function tree approach induces a lot of test cases of order n*5^(n-1). So, it is difficult to cope with such a large data and to check on these values which are more than 18k if n=6 and this results in exhaustive testing. While Divide-and-Rule approach is efficient but this method is limited to some extent e.g. it can solve Next Date problem and those like this problem. That is why a new method is required so that it can generalize the problem solving for input variables with functional dependencies.

CHAPTER – 3

3.1. Introduction

Software testing is one of the most widely and vividly used software analysis techniques in today’s world, one of which is Boundary Value Analysis (BVA). A modified version of BVA using input parameters with functional dependency is proposed in this chapter. The idea is derived from the inter dependency of functions among the input parameters. With this modified algorithm, an automated testing tool is created and implemented.

3.2. Implementation

In this chapter, three different models of BVA with respect to their functional dependencies are discussed and implemented. A comparison amongst them is made to find out the most suitable method as far as execution time.

3.3. Implementing General BVA

One of the three methods that is compared in this thesis is the general BVA. Boundary value analysis can be referred to as a technique or a method of a black box testing in which all the values as input at the boundaries of the domain of the input are tested. However, it has been widely recognized that the values at the extreme ends of the input, and may be just on the outside of, domains may tend to create significant amount of errors in system functionality.

This technique helps in boundary value testing which generally is done between both the valid and the invalid boundary partitions. With the help of this method or technique, all the boundary values are tested by creating the number of those test cases for a particular input field. This method does generate a very small number of test cases but it surely does not cover all the necessary cases as well. In general, this method generates 7*n number of test cases, with n being number of variables.

3.3.1 Example of BVA

Consider an organization that should test a software program which takes the values of the integers ranging between -100 to +100. Now in this case, total of three different sets of the valid equivalent partitions should be taken, which must be – the negative range which is from -100 to -1, the zero (0), and the positive range which is from 1 to 100.

All these ranges have a maximum and a minimum boundary value. A lower value of -100 and -1 as the upper value will be the negative range and the positive range will contain 1 as the lower value and the upper value as 100. Though, while testing these values, we should consider the fact that some of the values will overlap once the boundary values for each partition are selected. So, during the testing of these conditions, the overlapping values will appear when these boundaries being checked.

These values that have been overlapping must be removed so that the elimination of these redundant test cases can be done with ease. Now, after doing all that, the input box test cases that accepts the range of the integers between -100 and +100 through BVA are:

i All the test cases which have the same data as the input boundaries of input domain: in this case, -100 and +100.

ii All the test cases that have values just below the extreme edges of input domain: in this case, -101 and 99

iii All the test cases having values which are just above the extreme edges of input domain: in this case, -99 and 101

With the BVA technique, it is quite an easy task of testing such a small piece of data instead of doing a test on the whole data set. That is the reason, in major testing fields such as quality management services and software testing, this method is often adopted instead of other testing methods.

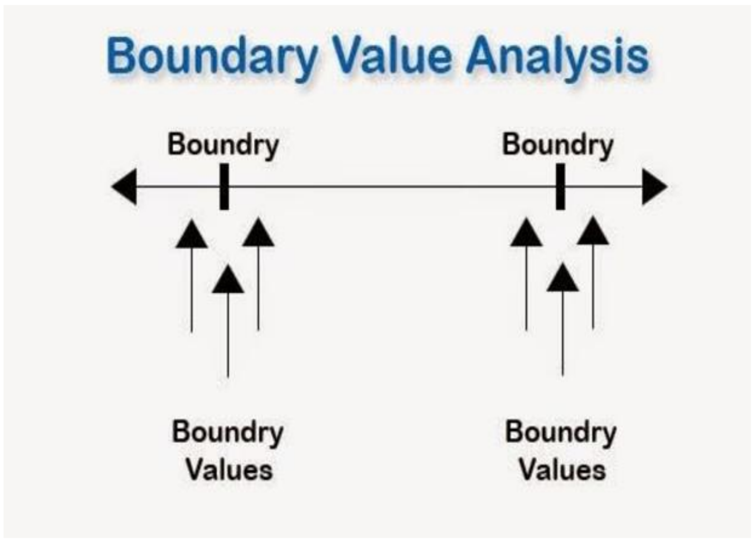


Fig 3.1: Boundary Value Analysis

3.3.2 BVA Analysis

For the generalized BVA method, the following are the advantages and the disadvantages:

Advantages are:

1. This technique is very good at exposing the potential user interface or user input problems
2. It has very clear guidelines on determining test cases
3. It is the best approach in all the cases in which the software functionality is based on numerous variables which generally represents the physical quantities.
4. It generates very small set of test cases

Disadvantages are:

1. It does not test all the possible inputs
2. It does not fit well with respect to the Boolean Variables
3. It generally works well only with those independent variables that usually depict quantity
4. It does not test the dependencies between combinations of inputs

Now, as this work is based on solving the problem of functional dependencies in the input variables, there is no scope for this generalized approach of boundary value analysis since this method does not deal with the dependencies.

3.4 Implementing Function Tree Algorithm

As discussed in Chapter 2, Function Tree Algorithm has in total three levels. Each level is an upgrade of the other. So, it's decided only to implement the last level i.e. Level 3. The reason for this choice is that this level is almost like the traditional boundary value analysis except that in traditional boundary value analysis, all except 1 has the middle value but in Level 3 of Function Tree Algorithm has the middle value in at-least one of these parameters.

Now, let assume that these three input parameters are X, Y and Z. Also, the following assumptions are made relating to the functions to be constrained:

(1) X_{min} will be the lower bound and X_{max} will be the upper bound for X. For Y, the upper bound and the lower bound will be the functions of X. So,

$Y_{max} = fU(X)$ ……………………………... equation 3.1

$Y_{min} = fL(X)$ ……………………………… equation 3.2

(2) For the third parameter, the upper bound and the lower bound will be functions of X and Y. So,

$Z_{min} = gL(X, Y)$ …………………………... equation 3.3

$Z_{max} = gU(X, Y)$ ………………………….. equation 3.4

(3) All the three parameters: X, Y and Z must satisfy the following constraint function:

$h(X, Y, Z) \geq 0$ or $h(X, Y, Z) = 0$ ……………. equation 3.5

Now, it is known in general that the test cases which might involve the boundary values are usually more important than those cases which involve “middle” values. Though, it does not necessary mean that the “middle” value is required for testing, especially when factors such as cost and time are permitted.

The test cases in this method are almost like the cases in the general boundary value analysis, taking two of the three parameters as middle values. The third parameters must go through the defined five elements sequence which is: min, min+, nom, max− and max. The most complete tree is the middle function tree. Every element has a middle function tree in the five elements sequence of the X’s that is: X_{min}, X_{min+}, X_{nom}, X_{max-}, and X_{max}.

Using this method, a total of five test cases can be generated from the left most path. So, in total, there are 25 test cases that are being generated from the Level 3 tree shown in Chapter 2. Though some of these test cases might not satisfy the Function Tree definition of the test cases. So, we will have to use constraint functions to remove all these unwanted or repetitive cases that do not satisfy the method’s conditions. However, if a test case satisfies the Function Tree definition of the test cases, it will be added to the test case list.

The Level 3 approach of the function tree method was originally inspired from the general geometry view point. Fig 3.2 shows a representation of the boundary value selection for each parameter in a geometry point of view.

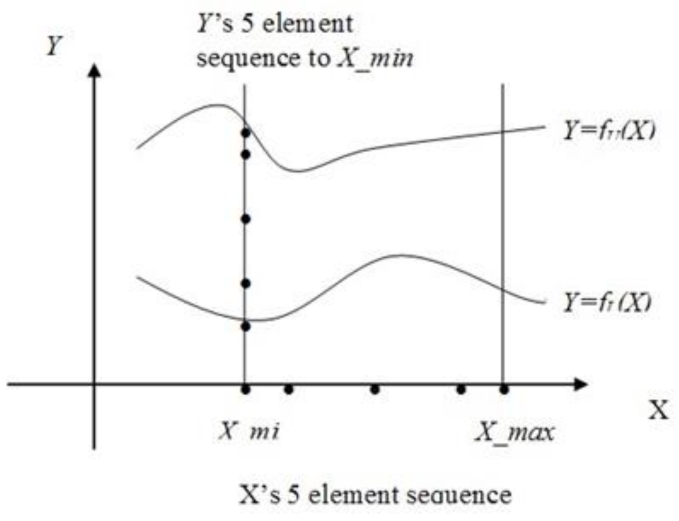


Fig 3.2: Geometry view of the implementation

An algorithm was devised for the automation testing tool. Note that due to complexity constraint, the algorithm caters for only two cases.

For two parameters (X, Y):

- Generate x's 5-element sequence x-lower, x-lower+ 1, x-middle, x-upper- 1, x-upper, where x-middle = (x-lower + x-upper) / 2.
- Loop through x in x-sequence to find the y's bounds.
- Generate y's 5-element sequence y-lower, y-lower+ 1, y-middle, y-upper - 1, y-upper.
- Loop through y in y-sequence. For each pair (x, y) if at least one element is a middle element then add it to the test cases list.

For three parameters (X, Y, Z):

- Generate x's 5-element sequence x-lower, x-lower+ 1, x-middle, x-upper- 1, x-upper.

- Loop through x in x-sequence to find the y’s bounds. Generate y’s 5-element sequence y-lower, y-lower + 1, y-middle, y-upper - 1, y-upper.
- Loop through y in y-sequence. For each pair (x, y) if it satisfies the constraints is In-Z-Constraints (x, y) find z’s lower and upper bounds.
- Generate z’s 5-element sequence z-lower, z-lower+ 1, z-middle, z-upper- 1, z-upper.
- For each triple (x, y, z) if at least one element is a middle element then add it to the test cases list.

Though, only algorithms for two cases are written i.e. two and three parameters but the function tree approach shown in this work is general and one can apply this approach to any constraints even with more than three parameters. Using this function tree algorithm, an automation testing tool was designed and implemented to decrease the time of testing.

3.5 Implementing the Proposed Approach

In the proposed approach, a new modified, based on boundary value analysis (BVA) algorithm works on the input parameters with functional dependencies being generated even with the invalid cases alongside valid cases to give more precise testing objectives.

Figure 3.5 represents the proposed modified method which works on the algorithm. Then, the variables defined will be designated its value. As you know, each variable will be given a boundary value that is, min as in minimum value of the range and max as in maximum value of the range. This summarizes the block number 2 of the block diagram.

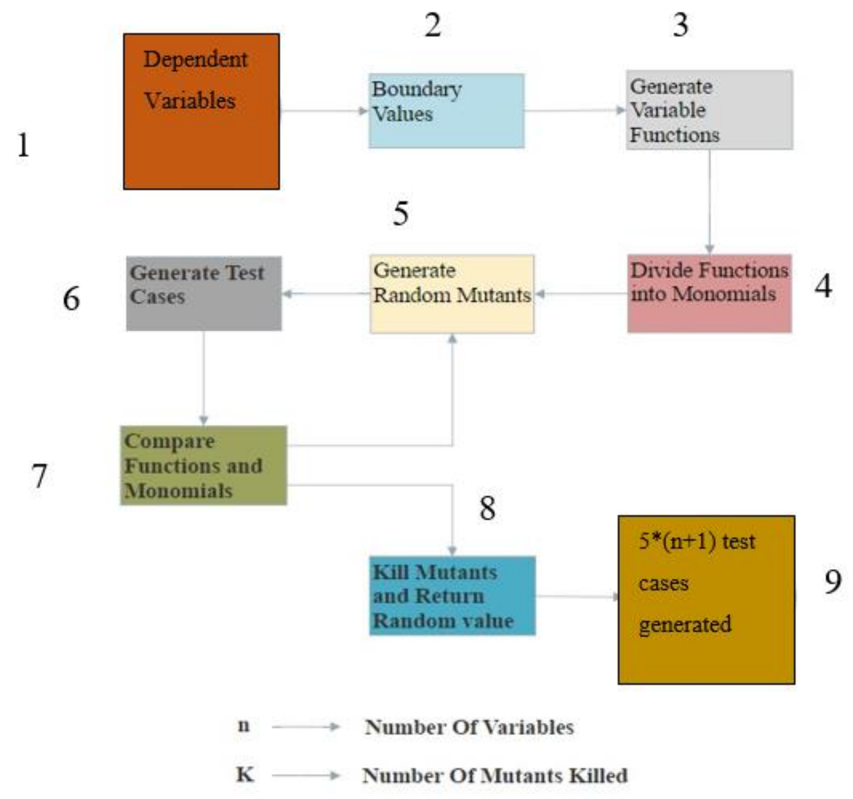

1
Dependent Variables
2
Boundary Values
3
Generate Variable Functions
4
Divide Functions into Monomials
5
Generate Random Mutants
6
Generate Test Cases
7
Compare Functions and Monomials
8
Kill Mutants and Return Random value
9
5*(n+1) test cases generated
n → Number Of Variables
K → Number Of Mutants Killed

Fig 3.4: Block Diagram of the modified approach

The following will explain the algorithm below in conjunction with fig.3.5 above, which represents the proposed modified method. For figure 3.5, starting with the stage number 1 and Step 1 in the algorithm by asking for variables details. First, it asks for the number of variables and their names. Then, we must define the number of dependent and independent variables. Though in this case, it is taken for granted that the first variable taken is the only independent variable and the other variables are dependent variables.

After designating the variable values, the algorithm moves to stage 3 in which various functions are created known as variable functions that are basically functions to represent the dependencies of the dependent variable. The values for the coefficients and the constant term are then entered for the function. For example, suppose x is the independent variable and y is the dependent variable. Then the function will look like:

$Y = f(X) = aX^4 + bX^3 + cX^2 + dX + e$ ……………………………… Equation 3.6

Depending on the function type that whether the function is linear or non-linear and if non-linear then what degree. These are various factors that concludes the creation of the function. According to equation 3.6, if the function is linear, then value of a, b, and c will be 0. Otherwise, the values of a, b, and c depends on the degree of the function and dependency.

Now, after creating these variable functions, they are divided into monomials by making them dependent on each variable one at a time. And if you are dividing a polynomial function by a monomial, it means that you are splitting the problem into different pieces. For instance, suppose if $Y = 3X^4 + 2X^2 +5$, then by dividing this polynomial function by X^2 and if we assume X^2 as Z, we can split the problem and make this polynomial function a monomial.

After dealing with the variable function, we will now generate random mutants in the function. For example, consider the following C++ code fragment**:**

```
 * This is a generic example to prove the case*
 * It is not from the code *
if (a && b) {
   c = 1;
}
else {
   c = 0;}
```

The condition mutation operator would replace && with || and produce the following mutant:

```
if (a || b) {
   c = 1;
}
else {
   c = 0;
}
```

This is how the mutants are generated. After we finish installing a mutant at every step of the monomial, several test cases are generated. Now these test cases are generated according to the traditional boundary value analysis methods or techniques.

Now, with these mutants generated at every step, we will compare the values of the functions as well as the monomials also at every step respect to the mutation and the test case. If the value at the variable function is equal to the value at the monomials, then we should repeat the step of mutation and test case generation again because it means mutant is still there and nothing happened to it. Otherwise, we will follow the next step because mutant is killed.

As soon as the mutant is killed, the random value used to generate the test cases and the mutants is chosen and returned to be the kth test case where k being the number of mutants killed. So, this is how we create mutants and kill them to generate test cases. This concludes the step 6 of the algorithm.

Now, all the test cases are generated which are required or necessary to generate by an easy and efficient automation testing method using the mutation testing.

3.5.1 Modified Algorithm:

1) First 7*n test cases are computed by applying the traditional approach on the matrix??? created by Level 3 algorithm.
2) The dependent variable functions or polynomials are generated.
3) These functions are divided into monomials by making them dependent on each variable one at a time.
4) Generate test cases by randomly creating the mutants at every step.
5) If the computed value of the F(Polynomial) is equal to the computed monomial value then repeat Step 4 and Step 5 else Mutant is killed.

6) Return the random value as soon as the mutant is killed being the 5*n +kth test case, with K being the number of mutants killed up to this point.

Now, in the algorithm given in chapter 2 i.e. the Function tree, there are five values for every parameter min, min+1, nom, max-1 and max. However, a modification is made by the inclusion of two parameters i.e. min-1 and max+1 thereby giving the advantage of testing on the invalid cases.

In level 3 of the Function tree i.e. the Normal Test in which at-least one of the parameters should be nom, others now can have seven different values which will make the testing objective more precise. The seven values considered as parameters are min-1, min, min+1, nom, max-1, max and max+1. For instance, X, Y and Z are the three parameters with Y = f (X) and Z = f (X, Y). There are seven boundary values for every parameter.

Now assign all but one parameter the middle value, i.e. nom, and that parameter can have any value of those seven. So, X can have any of the seven values then Y and Z can only have the nom value out of those seven. So, the total combinations for X = 7 and again same goes with Y and Z. So, the total number of possible combinations for three input parameters with functional dependency comes out to be 7*3 i.e. 21. These test cases include invalid cases for testing too. So, in general total number of possible test cases for n input parameters with functional dependencies are:

# Test Cases = (n*7).

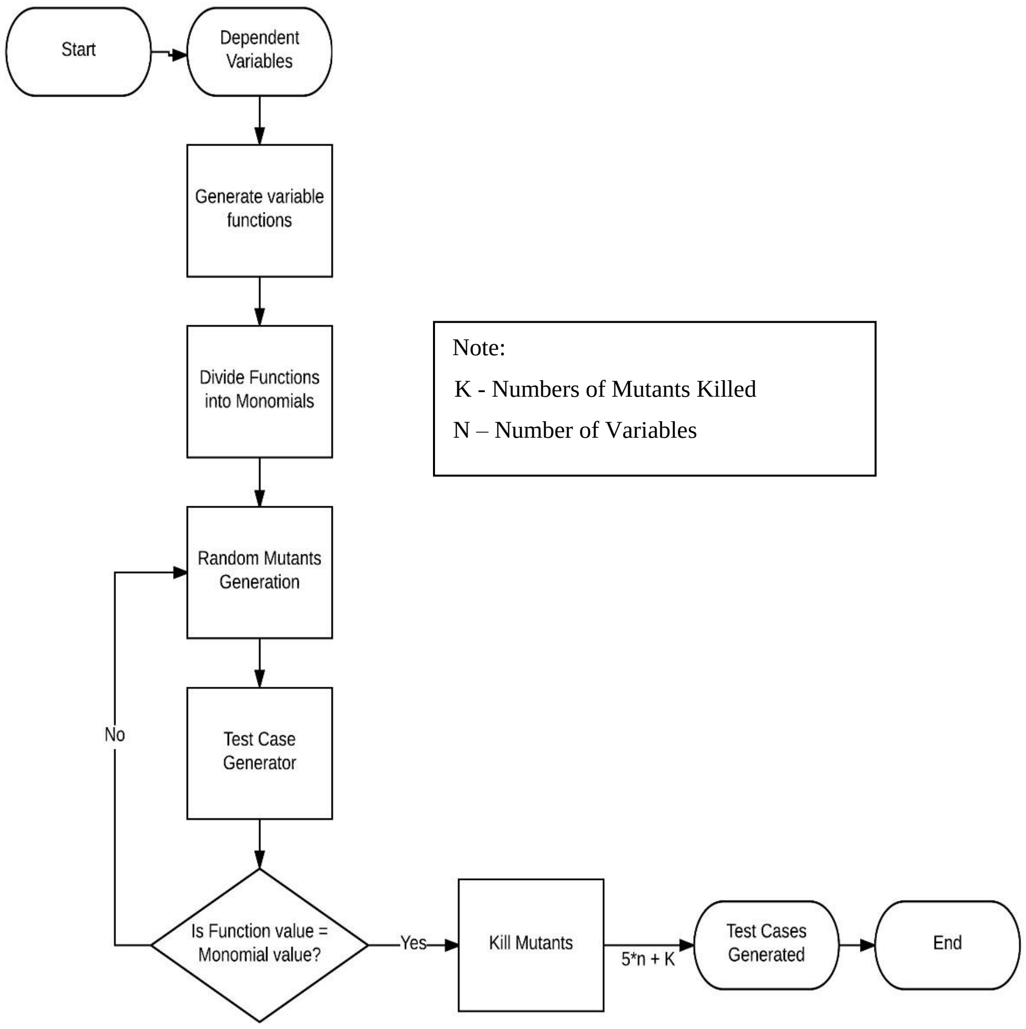

Start
Dependent Variables
Generate variable functions
Divide Functions into Monomials
Random Mutants Generation
Test Case Generator
Is Function value = Monomial value?
No
Yes
Kill Mutants
5*n + K
Test Cases Generated
End
Note:
K - Numbers of Mutants Killed
N – Number of Variables

Fig 3.3: Flowchart of the new modified method

Next Step in the algorithm is to divide the polynomial function into subsequent monomial function for each dependent variable. The test cases are generated by randomly creating the mutants at every step. If the computed value of F(Polynomial) is not equal to computed monomial mutant value, then generate this random number as the kth test case else repeat the previous step 50 times, which is an arbitrary number, till the condition is satisfied.

For instance, if Z= f (X, Y) such that Z=2X+3Y. Now Z is divided into two functions, 2X and 3Y. Basically the coefficient of the other parameter is reduced to 0. This will create mutants in the program causing a deliberate fault in the algorithm to check whether the test cases are still working or not. In this example if X=2 and Y=3 is given then Z will be either 4 or 9 but it should give the output 13. So, the test cases should not work due to the mutant present in it. So, the algorithm randomly creating and killing the mutant before some unknown error can create a mutant by mistake. This will reduce the unwanted errors.

CHAPTER – 4

4.1. Introduction

In the previous chapter, different algorithms, and different types of techniques on how software technology in terms of testing is improving largely.

In this chapter, the difference amongst all the implementations in technical terms as well as their result are discussed namely Traditional approach, Function Tree approach, and the proposed approach the results of which are analyzed, and compared. Note that chapter 4 test results are based on 30 different cases, values, and different number of variables, however, only few cases are shown.

4.2. Analyzing General Boundary Value Analysis

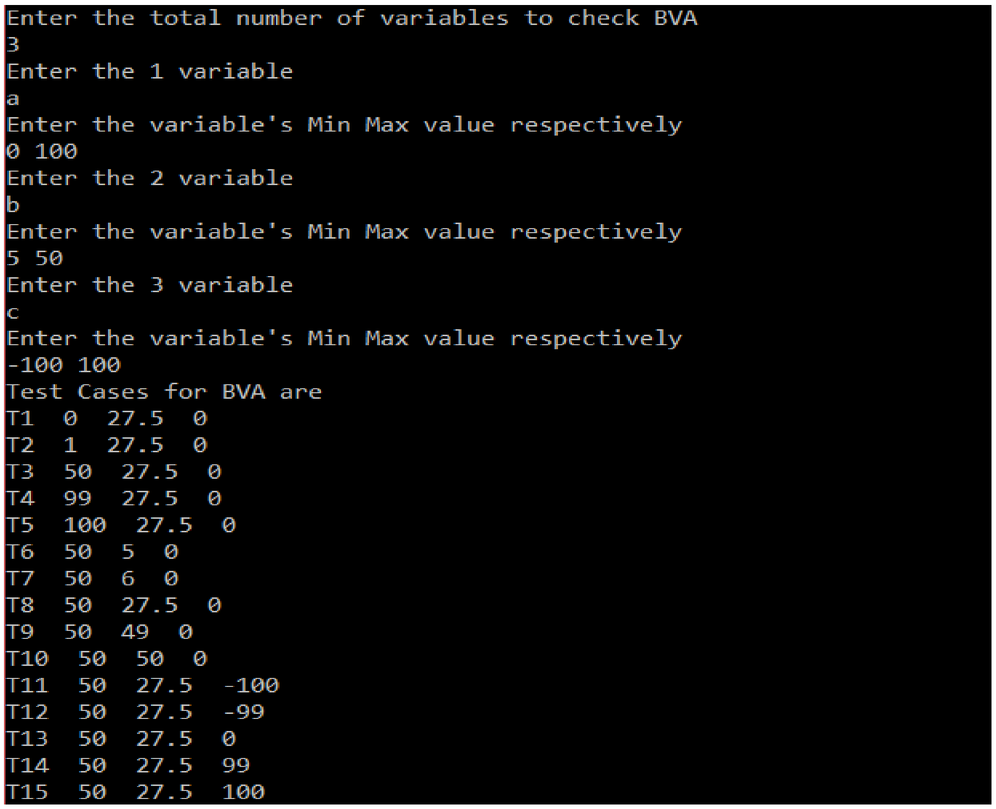
The following are made for the implementation of the generalized BVA.

- Enter the number of variables to be tested.
- Enter the name of those variables.
- Enter the Min and the Max value of those variables.

Also, for the above implementation, the following assumptions are made:

- It does not check for the values outside the range of Min-Max.
- There are no dependent variables.
- It largely depends on the Mid-value or the average value.

Case 4.2.1:



```
Enter the total number of variables to check BVA
3
Enter the 1 variable
a
Enter the variable's Min Max value respectively
0 100
Enter the 2 variable
b
Enter the variable's Min Max value respectively
5 50
Enter the 3 variable
c
Enter the variable's Min Max value respectively
-100 100
Test Cases for BVA are
T1  0  27.5  0
T2  1  27.5  0
T3  50  27.5  0
T4  99  27.5  0
T5  100  27.5  0
T6  50  5  0
T7  50  6  0
T8  50  27.5  0
T9  50  49  0
T10  50  50  0
T11  50  27.5  -100
T12  50  27.5  -99
T13  50  27.5  0
T14  50  27.5  99
T15  50  27.5  100
```

Fig 4.1: Result of Case 4.2.1

Number of Variables: 3

Name of the variables: a, b, c

Min & Max value of a: 10, 100

Min & Max value of b: 5, 50

Min & Max value of c: -100, 100

Total Number of test cases generated: 15

Figure 4.2, below, shows the actual number of test cases versus number of variables that are made throughout this implementation.

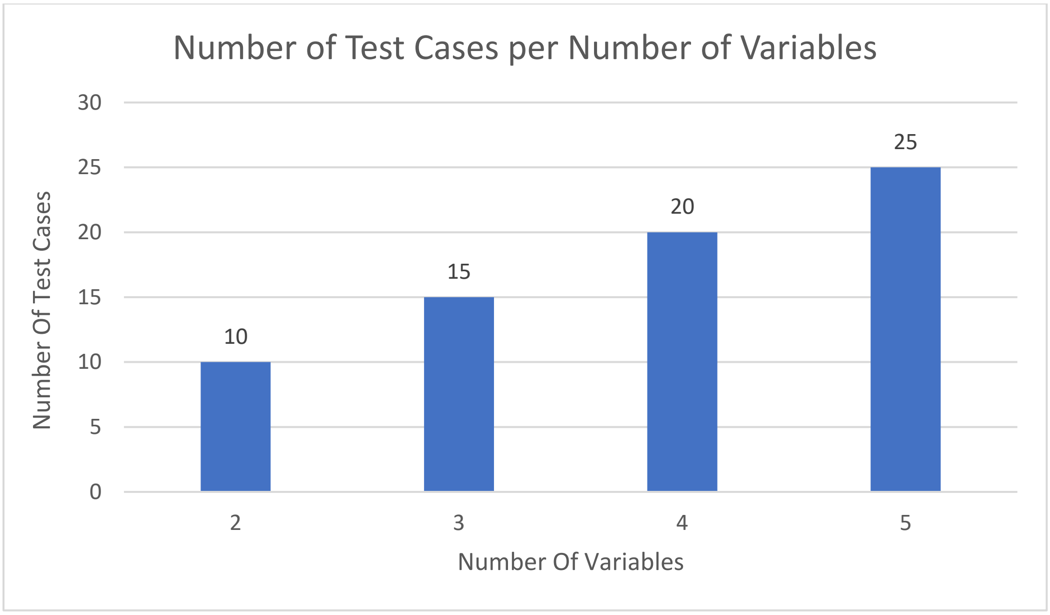


Fig 4.2: Number of Test Cases per Number of Variables in General BVA

In general, this technique generates 5*n number of test cases where n is the number of variables. Five different cases being min, min+1, mid, max-1 and max.

The dawn side of this method is that it does not solve the problem of dependency while the task for presented work is based on solving the BVA for input variables with functional dependency.

4.3. Analyzing Function Tree Approach

As compared with general BVA above, this method will deal with functional dependency and it is based upon the following;

- To check dependency, it is required to enter at-least 2 variables.
- The first variable to enter is always independent as the dependent variable needs a variable to depend upon as well.

- The highest degree considered in the equation of the dependent variable is three because of the complexity issue.

Also, the following are the steps for method implementation:

- Enter the number of variables to be tested.
- Enter the name of the variables.
- Enter the Min and Max of the first variable

  [Note: 1st variable is assumed to be independent]
- Enter the coefficients based on the nature of variable's degree.

For computational complexities, the highest degree considered is three. If the variable is dependent on the other variable linearly, then the coefficients of other degrees in the equations are made zero. The equations that are being created are there to measure the quantity of dependency or may be to tell the degree of dependency of one variable to the other.

<u>Case 4.3.1:</u>

Number of Variables: 2

Name of the variables: a, b

Min & Max value of a: 14, 67

Equation for F(b):

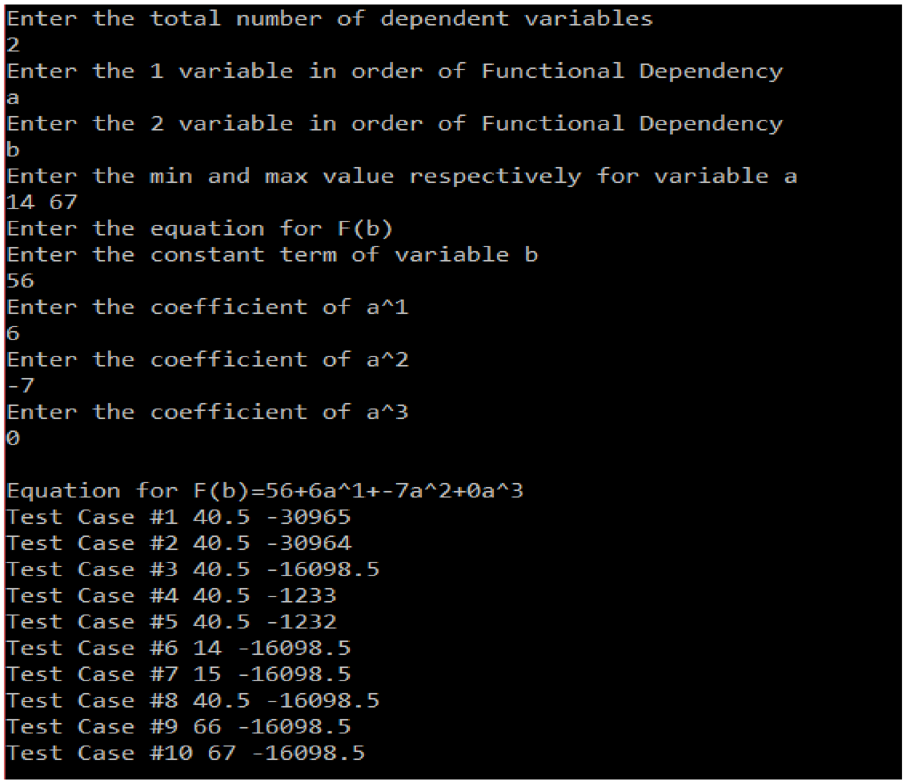
Constant Term: 56

Coefficient of a^1: 6

Coefficient of a^2: -7

Coefficient of a^3: 0

Total Number of test cases generated: 10



```
Enter the total number of dependent variables
2
Enter the 1 variable in order of Functional Dependency
a
Enter the 2 variable in order of Functional Dependency
b
Enter the min and max value respectively for variable a
14 67
Enter the equation for F(b)
Enter the constant term of variable b
56
Enter the coefficient of a^1
6
Enter the coefficient of a^2
-7
Enter the coefficient of a^3
0

Equation for F(b)=56+6a^1+-7a^2+0a^3
Test Case #1 40.5 -30965
Test Case #2 40.5 -30964
Test Case #3 40.5 -16098.5
Test Case #4 40.5 -1233
Test Case #5 40.5 -1232
Test Case #6 14 -16098.5
Test Case #7 15 -16098.5
Test Case #8 40.5 -16098.5
Test Case #9 66 -16098.5
Test Case #10 67 -16098.5
```

Fig 4.3: Results of Case 4.3.1

Case 4.3.2:

Number of Variables: 3

Name of the variables: a, b, c

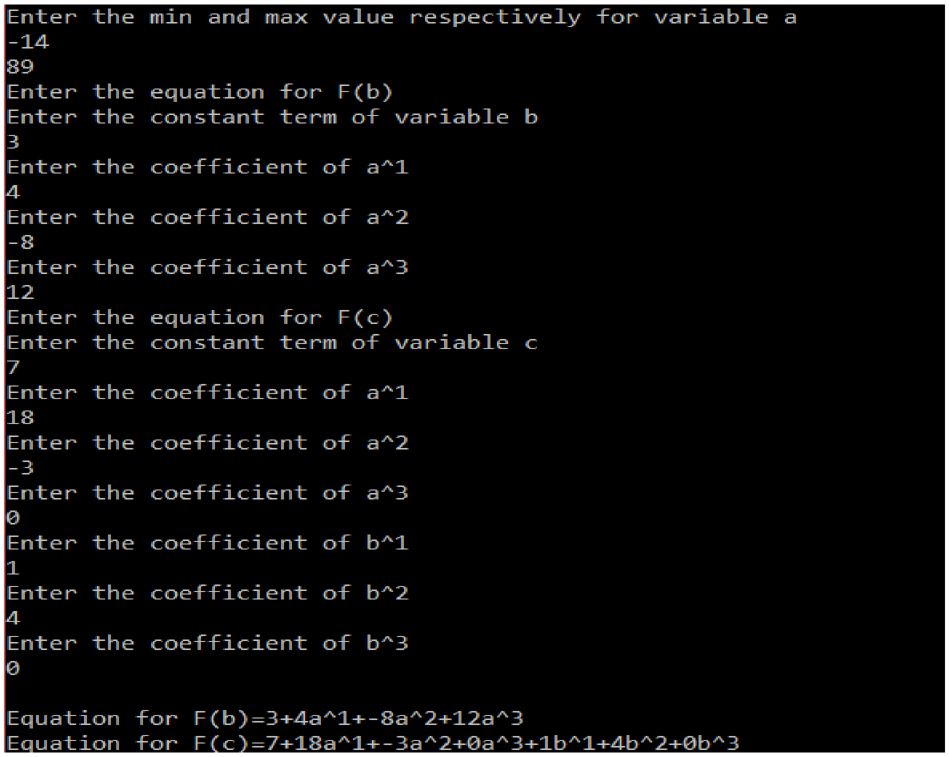
Min & Max value of a: -14, 89

Equation for F(b):

$3+4a-8a^2+12a^3$

Equation for F(c):

$7+18a-3a^2+b+4b^2$


```
Enter the min and max value respectively for variable a
-14
89
Enter the equation for F(b)
Enter the constant term of variable b
3
Enter the coefficient of a^1
4
Enter the coefficient of a^2
-8
Enter the coefficient of a^3
12
Enter the equation for F(c)
Enter the constant term of variable c
7
Enter the coefficient of a^1
18
Enter the coefficient of a^2
-3
Enter the coefficient of a^3
0
Enter the coefficient of b^1
1
Enter the coefficient of b^2
4
Enter the coefficient of b^3
0

Equation for F(b)=3+4a^1+-8a^2+12a^3
Equation for F(c)=7+18a^1+-3a^2+0a^3+1b^1+4b^2+0b^3
```

Fig 4.4: Inputs for Test Case 4.3.2

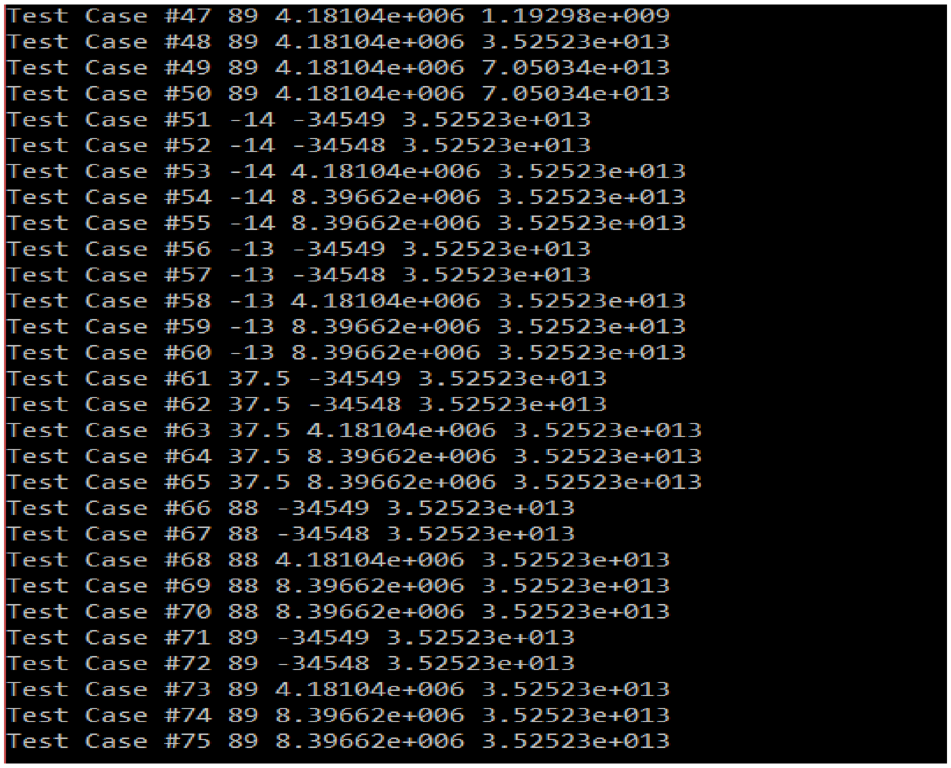

```
Test Case #47 89 4.18104e+006 1.19298e+009
Test Case #48 89 4.18104e+006 3.52523e+013
Test Case #49 89 4.18104e+006 7.05034e+013
Test Case #50 89 4.18104e+006 7.05034e+013
Test Case #51 -14 -34549 3.52523e+013
Test Case #52 -14 -34548 3.52523e+013
Test Case #53 -14 4.18104e+006 3.52523e+013
Test Case #54 -14 8.39662e+006 3.52523e+013
Test Case #55 -14 8.39662e+006 3.52523e+013
Test Case #56 -13 -34549 3.52523e+013
Test Case #57 -13 -34548 3.52523e+013
Test Case #58 -13 4.18104e+006 3.52523e+013
Test Case #59 -13 8.39662e+006 3.52523e+013
Test Case #60 -13 8.39662e+006 3.52523e+013
Test Case #61 37.5 -34549 3.52523e+013
Test Case #62 37.5 -34548 3.52523e+013
Test Case #63 37.5 4.18104e+006 3.52523e+013
Test Case #64 37.5 8.39662e+006 3.52523e+013
Test Case #65 37.5 8.39662e+006 3.52523e+013
Test Case #66 88 -34549 3.52523e+013
Test Case #67 88 -34548 3.52523e+013
Test Case #68 88 4.18104e+006 3.52523e+013
Test Case #69 88 8.39662e+006 3.52523e+013
Test Case #70 88 8.39662e+006 3.52523e+013
Test Case #71 89 -34549 3.52523e+013
Test Case #72 89 -34548 3.52523e+013
Test Case #73 89 4.18104e+006 3.52523e+013
Test Case #74 89 8.39662e+006 3.52523e+013
Test Case #75 89 8.39662e+006 3.52523e+013
```

Fig 4.5: Results of Case 4.3.2

Figure 4.5 below, shows the actual number of test cases versus number of variables that are made throughout this implementation.

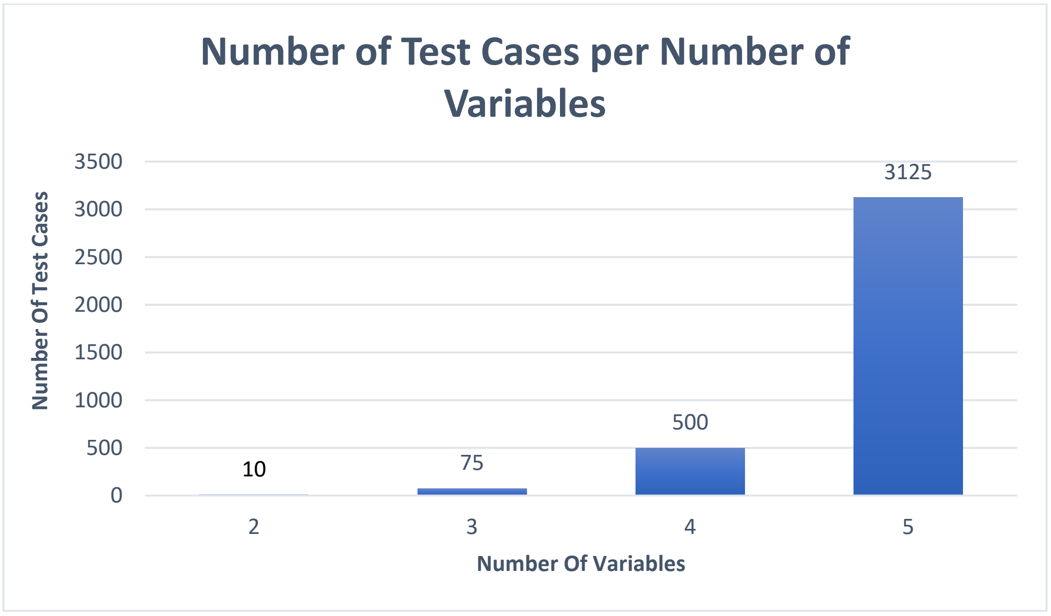


Fig 4.6: Number of Test Cases per Number of Variables in the Function Tree Approach

As shown in fig. 4.5, in the case of 2 variables, total number of test case generated are 10. Similarly, in the case of 3 variables, 75 test cases are being generated and similarly in the case of 4 variables, 500 test cases are being generated and so on. In general, this technique generates $n*5^{(n-1)}$ number of test cases where n is the number of variables. Five different cases being min, min+1, mid, max-1 and max. This approach theoretically gives significantly better results in terms of efficiency but produce more test cases.

4.4. Analyzing the Modified Approach

In the previous section, the Function Tree approach was analyzed through which it was found that the approach provides overwhelming test cases to deal with. In this section, the proposed method is to deal with the problem of functional dependency efficiently with an average number of test cases.

The inputs used to implement this modified approach is almost like the Function Tree approach.

- Enter the number of variables to be tested.
- Enter the name of the variables.
- Enter the Min and Max of the first variable
- Enter the coefficients based on the nature of variable's degree.

Case 4.4.1:

Number of Variables: 3

Name of the variables: a, b, c

Min & Max value of a: 1, 10

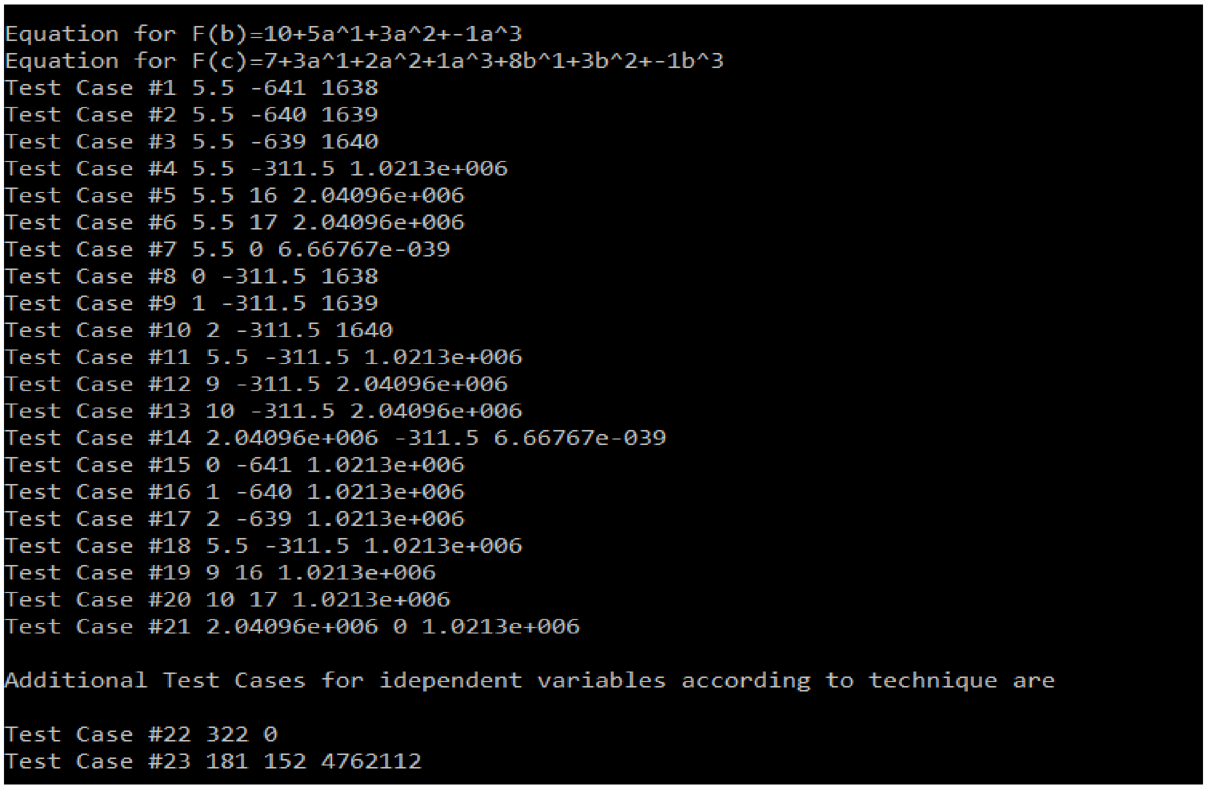
Equation for F(b):

$10+5a+3a^2-a^3$

Equation for F(c):

$7+3a+2a^2+a^3+8b+3b^2-b^3$

Fig 4.7 shows that total number of test cases for "Case 4.4.1" in which only three variables are used generates 21 regular test cases and 2 additional test cases.



```
Equation for F(b)=10+5a^1+3a^2+-1a^3
Equation for F(c)=7+3a^1+2a^2+1a^3+8b^1+3b^2+-1b^3
Test Case #1 5.5 -641 1638
Test Case #2 5.5 -640 1639
Test Case #3 5.5 -639 1640
Test Case #4 5.5 -311.5 1.0213e+006
Test Case #5 5.5 16 2.04096e+006
Test Case #6 5.5 17 2.04096e+006
Test Case #7 5.5 0 6.66767e-039
Test Case #8 0 -311.5 1638
Test Case #9 1 -311.5 1639
Test Case #10 2 -311.5 1640
Test Case #11 5.5 -311.5 1.0213e+006
Test Case #12 9 -311.5 2.04096e+006
Test Case #13 10 -311.5 2.04096e+006
Test Case #14 2.04096e+006 -311.5 6.66767e-039
Test Case #15 0 -641 1.0213e+006
Test Case #16 1 -640 1.0213e+006
Test Case #17 2 -639 1.0213e+006
Test Case #18 5.5 -311.5 1.0213e+006
Test Case #19 9 16 1.0213e+006
Test Case #20 10 17 1.0213e+006
Test Case #21 2.04096e+006 0 1.0213e+006

Additional Test Cases for idependent variables according to technique are

Test Case #22 322 0
Test Case #23 181 152 4762112
```

Fig 4.7: Results of Case 4.4.1

<u>Case 4.4.2:</u>

Number of Variables: 4

Name of the variables: a, b, c, d

Min value of a: -1

Max value of a: 10

Equation for F(b):

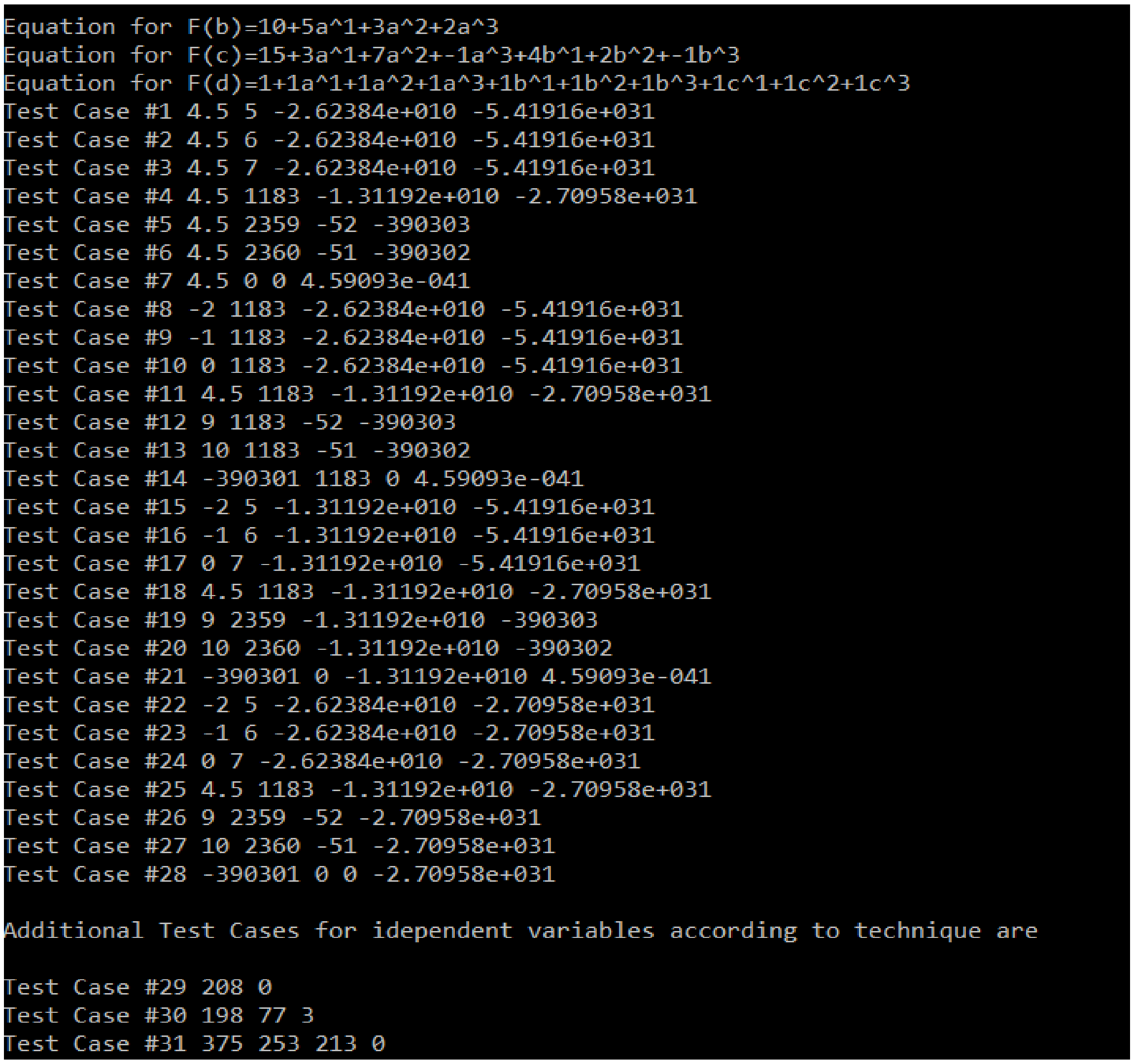
$10+5a+3a^2+2a^3$

Equation for F(c):

$15+3a+7a^2-a^3+4b+2b^2-b^3$

Function for F(d):

$1+a+a^2+a^3+b+b^2+b^3+c+c^2+c^3$



```
Equation for F(b)=10+5a^1+3a^2+2a^3
Equation for F(c)=15+3a^1+7a^2+-1a^3+4b^1+2b^2+-1b^3
Equation for F(d)=1+1a^1+1a^2+1a^3+1b^1+1b^2+1b^3+1c^1+1c^2+1c^3
Test Case #1 4.5 5 -2.62384e+010 -5.41916e+031
Test Case #2 4.5 6 -2.62384e+010 -5.41916e+031
Test Case #3 4.5 7 -2.62384e+010 -5.41916e+031
Test Case #4 4.5 1183 -1.31192e+010 -2.70958e+031
Test Case #5 4.5 2359 -52 -390303
Test Case #6 4.5 2360 -51 -390302
Test Case #7 4.5 0 0 4.59093e-041
Test Case #8 -2 1183 -2.62384e+010 -5.41916e+031
Test Case #9 -1 1183 -2.62384e+010 -5.41916e+031
Test Case #10 0 1183 -2.62384e+010 -5.41916e+031
Test Case #11 4.5 1183 -1.31192e+010 -2.70958e+031
Test Case #12 9 1183 -52 -390303
Test Case #13 10 1183 -51 -390302
Test Case #14 -390301 1183 0 4.59093e-041
Test Case #15 -2 5 -1.31192e+010 -5.41916e+031
Test Case #16 -1 6 -1.31192e+010 -5.41916e+031
Test Case #17 0 7 -1.31192e+010 -5.41916e+031
Test Case #18 4.5 1183 -1.31192e+010 -2.70958e+031
Test Case #19 9 2359 -1.31192e+010 -390303
Test Case #20 10 2360 -1.31192e+010 -390302
Test Case #21 -390301 0 -1.31192e+010 4.59093e-041
Test Case #22 -2 5 -2.62384e+010 -2.70958e+031
Test Case #23 -1 6 -2.62384e+010 -2.70958e+031
Test Case #24 0 7 -2.62384e+010 -2.70958e+031
Test Case #25 4.5 1183 -1.31192e+010 -2.70958e+031
Test Case #26 9 2359 -52 -2.70958e+031
Test Case #27 10 2360 -51 -2.70958e+031
Test Case #28 -390301 0 0 -2.70958e+031

Additional Test Cases for idependent variables according to technique are

Test Case #29 208 0
Test Case #30 198 77 3
Test Case #31 375 253 213 0
```

Fig 4.8: Results of Case 4.4.2

Total number of test cases generated in this case are 28 + 3 = 31.

In general, this technique generates (7*n) + k number of test cases where n is the number of variables and k is the number of mutants killed. Seven different cases being min-1, min, min+1, mid, max-1, max, and max+1.

This approach theoretically gives significantly better results in terms of efficiency compared to the regular BVA technique but provides average amount of test cases. In some scenarios, this method is much better than the traditional BVA method as well the Function Tree method.

4.5. Comparison Amongst General BVA, Function Tree, and Modified Approach

Fig 4.9 shows the outcome achieved through this work. The modified method shows good improvement in lowering the number of test cases, with much more improvements as the number of input variables increases.

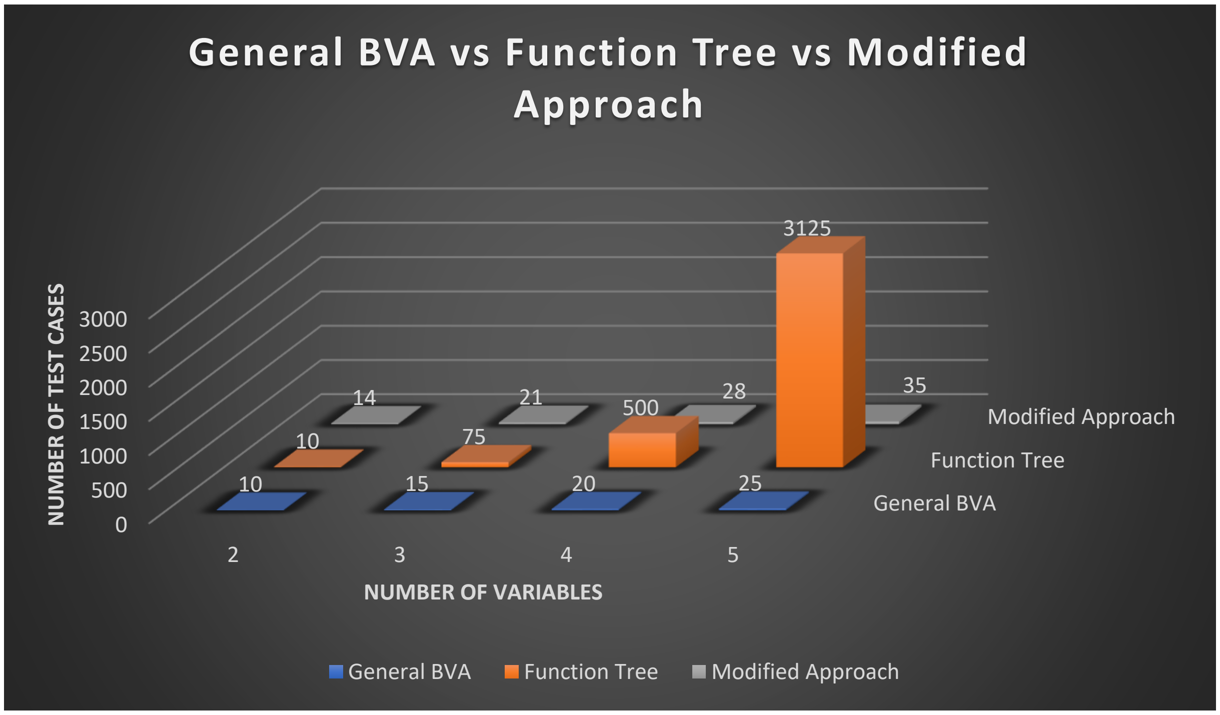


Fig 4.9: General BVA vs Function Tree vs Modified Approach

The proposed method will evaluate almost every possible required test case increasing the system's efficiency.

	Generalized BVA	**Function Tree**	**Modified BVA**
Number of Test Cases	5*n	n*5^(n-1)	(7*n) + k
Mutation Score	Least	Lesser than Modified BVA	More than both Generalized BVA and FTA
Equivalent Mutants > 0	Works	Works	Doesn't work
Number of Variables	Works with finite number of variables	Difficult to implement if n > 4	Works with finite number of variables
Range Check	Between min and max	Between min and max	Checks for entire range

Table 1: Generalized BVA vs Function Tree Algorithm vs Modified Algorithm

The table 1 shows a comparison amongst the generalized BVA method, Function Tree approach, and the modified method. In table 1, the red marking of the modified approach shows how it is better in some ways than the other two methods compared with. In the modified BVA, mutation testing is also included due to which mutation score is counted and the table is created according to the hypothetical inclusion of the mutation in other two techniques.

The elements of the Mutation Score (M_S) are;

D = Dead Mutants;

N = Number of Mutants;

E = Equivalent Mutants;

Where;

M_S= (D / (N-E)) ----------------------------Eq. 4.1

Mutation Score will be significantly more than the other two approaches because the implementation of this modified method is done considering there are no Equivalent Mutants. If there are equivalent mutants, the modified BVA will not recognize it and will fail to work on it while other will work in the case of Equivalent Mutants.

Now, moving on to the numbers of test cases and variables. Generalized BVA generates 5*n number of test cases while Function Tree method generates the highest of all three i.e. n*5^(n-1) and the modified approach generates 7*n + k number of test cases where n is number of variables and k being the number of mutants killed at each step. Both generalized BVA and the proposed technique (modified BVA) works well with finite number of variables which can be 5, 10, 20 etc. But in the case of Function tree algorithm, it only works till 4 or 5 variables, since the number of test cases will become so large that it will be impossible to test and will therefore encounter complexity issues.

CHAPTER 5

5.1. Conclusion

There are several techniques of Software Testing in this world but people do not know which technique to use and when. So, this work is a help to understand testing a little better. One of which is Boundary Value Analysis. A modified version of Boundary Value Analysis using input parameters with functional dependency is proposed in this work. The idea is derived from the inter dependency of functions among the input parameters. With this modified algorithm, an automated testing tool is created and implemented.

Generalized BVA generates 5*n number of test cases where n is number of variables while Function Tree method generates the highest of all three that is n*5^(n-1) and the modified approach generates 7*n + k number of test cases where k is the number of mutants killed at each step. So, it shows that the number of test cases in case of modified algorithm is significantly lower than the Function Tree algorithm while almost similar as regular BVA but it covers more functionalities and features as shown in Chapter 4.

This testing tool shows the advantages of the modified algorithm developed over the Functional Tree Approach and reduces a significant amount of test cases that leads to an exhaustive testing. This modified method will test almost every possible required test case increasing the system’s efficiency compared to regular

BVA in terms of speed and number of test cases. This method will be a very good help for any product based company saving some amount of money and time.

5.2. Future Work

In this work, the Function tree approach induces a lot of test cases of order n*5^(n-1). So, it is difficult to cope with such a large data and to check on these values which are more than 18k if n=6 and this results in exhaustive testing. Thus, the proposed algorithm reduces the test cases significantly to 7*n +k; k being the killed mutants.

Though the algorithm requires at-least one independent variable so problems like triangle inequality can't be handled. So, for the future algorithm can be further modified to incorporate the case when all variables are interdependent.

REFERENCES

[1] T.Y.Chen, M.Y.Cheng, P.L.Poon, T.H.Tse, and Y.T.Yu, “A study of input domain partitioning”, Proceedings of the 20th IASTED International Multi-Conference on Applied Informatics (AI 2002), ACTA Press, Calgary, Canada, pp. 176-181, 2002.

[2] Karan Vij and Wenying Feng, “Boundary Value Analysis using Divide- and –rule Approach”, Fifth International Conference on Information and Technology, 2008, IEEE, pg. 70-75.

[3] WENYING FENG," A Generalization of Boundary Value Analysis for Input Parameters with Functional Dependency",9th IEEE/ACIS International Conference on Computer and Information Science, 2010, IEEE, pg. 776-782.

[4] Guru99 2017, Boundary Value Analysis & Equivalence Partitioning with Examples, accessed 14 March 2017, <http://www.guru99.com/equivalence-partitioning-boundary-value-analysis.html>.

[5] Stack Exchange, Functional dependency and normalization, accessed 19 March 2017, <http://stackoverflow.com/questions/4199444/functional-dependency-and-normalization>.

[6] J. C. Cherniavsky, “Validation, Verification, and Testing of Computer Software”, National Institute of Standards and Technology (NIST), February 1981. NIST 500-75.

[7] Ali Parsai, Alessandro Murgia, Quinten David Soetens, and Serge Demeyer, “Mutation Testing as a Safety Netfor Test Code Refactoring”, Research Gate, 2015

[8] Ranorex GmbH, Test Automation and its Benefits, accessed 15 March 2017, <http://www.ranorex.com/why-test-automation.html>.

[9] QATestLab, Automated Testing, accessed 15 March 2017, <http://qatestlab.com/services/we-are-professionals-in/automated-testing/>.

[10] N. Debnath and J. R. Haakenson, “Automated testing design and description for certain functions”, Proceedings of the Sixth IEEE International Conference on Electro/Information Technology, (IEEE-EIT 2006), Michigan State University, East Lansing, MI, USA, May 7-10, 2006.

[11] W. Feng and Z. Zhang, “Sequence algorithms for boundary value analysis with constrained input parameters”, Proceedings of the ISCA 14th International Conference on Intelligent and Adaptive Systems and Software Engineering (IASSE-2005), Toronto, Canada, pp. 255-260, 2005.

[12] N. Debnath, W. Feng, M. Burgin, J. Wilson, and J. Cropper: “An auto tester for the modified Next Date function reusing a test case generator”, Proceedings of the IEEE International Conference on Information Reuse and Integration (IEEE IRI-2004), Las Vegas, USA, pp.121126, 2004.

[13] R. Pressman, Software Engineering - A Practitioner’s approach, 5th Edition, The McGraw Hill Companies, Inc., New York, 2001.