AUTOMATED TEST DATA GENERATION USING HEURISTIC TECHNIQUES

Arindam Saha
Department of Computer Science & Engineering
Girijananda Chowdhury Institute of Management and Technology
Guwahati, Assam -781017
arindamcse@gimt-guwahati.ac.in

Keywords: SUT, ATDG, Control Flow Graph, Test Data.

ABSTRACT

Automated Test Data Generation (ATDG) is an activity that in the course of software testing automatically generates test data for the software under test (SUT). It usually makes the testing more efficient and cost effective. Test Data Generation (TDG) is crucial for software testing because test data is one of the key factors for determining the quality of any software test during its execution. The multi-phased activity of ATDG involves various techniques for each of its phases. But till now a fully automated system is not available which will provide all the benefits.

INTRODUCTION

Automatic Test Data Generation Technique (ATDG) is a technique to automatically generate test data for ***Software under Test (SUT).*** It is an activity that in the course of software testing automatically generates test data for the software under test. It makes software testing more efficient and cost effective. ATDG is crucial because test data is one of the crucial factors that determine the quality of a software test during its execution. ATDG reduces testing time by and order of magnitude and presses the rising cost of test data generation downwards. All new inventions are parsed through a variety of tests which are based on a well-established criterion developed over the years. Mostly this activity is generated to verify the performance and quality of the product.

Generally software is tested for its structure or functions which are supposed to be performed by its components .Testing in the prior is commonly known as structural or white-box testing and in the later case it is known as functional or black-box testing. For either of the cases, test data is required to “traverse” through the SUT. The outcome of the traversal determines correctness, performance, or in general, the quality of the software.

Heuristic Technique refers to experience-based techniques for problem solving, learning and discovery. Heuristic methods are used to speed up the process of finding a satisfactory solution, where an exhaustive search is impractical. Examples of this method include rule of thumb method, an educated guess, an intuitive judgment or just common sense. In precise terms, heuristics are strategies that are readily accessible, though loosely applicable information to control problem solving in humans and machines.

1.1 **Need of automation in testing**

Unreliable software is considered to be prone to failures and hence, in general, carries little worth. Software testing which is the "process of ensuring that a certain piece of software item fulfills its requirements" is one of the vital factors that can ensure reliability of the software. Assurance of software reliability partially depends on testing. However it is interesting to note that testing itself also needs to be reliable. Automating the testing process is a sound engineering approach, which can make the testing efficient, cost effective and reliable. Further mostly customers rely on software that is prepared according to some authenticated standards. Automization could promote standardization through standardized and patent test data generation tools.

Automated test data generation is one of the core factors that contribute towards automated testing. It is used for automatically generating test data for a SUT during software testing. Test data generation is a tedious, expensive and error prone process, if it is done manually. Automated software testing increases test coverage. Even the most conscientious tester will make mistakes during monotonous manual testing. Automated tests perform the same steps precisely every time they are executed and never forget to record detailed results. Moreover, automated testing can simulate tens, hundreds or thousands of virtual users interacting with network or web software and applications.

Thus, automation in the test data generation process could curtail testing expenses and at the same time, increase the reliability of testing as a whole. For these reasons automated test data generation has remained a topic of interest for the past four decades.

1.2 **Types of automation in testing**

ATDG itself is a multi-phased process, which involves different techniques for every phase. The different ways to generate test data are:-

a) Random test data generation
b) Goal oriented test data generation
c) Path Oriented test data generation
d) Heuristic technique

a) Random test data generation:

Random test data generation simply consists of generating inputs at random until a useful input is found. This approach is quick and simple but might be a poor choice

with complex programs and with complex adequacy criteria. The probability of selecting an adequate input by chance could be low in this case.

The biggest issue for random approach is that of adequate test data selection.

b) Goal oriented test data generation:

In the goal-oriented approach, test-data is selected from the available pool of candidate test data to execute the selected goal, such as a statement, irrespective of the path taken. This approach involves two basic steps: to identify a set of statements (respective branches) the covering of which implies covering the criterion; to generate input test data that execute every selected statement (respective branch).

Generally the goal-oriented approach faces issues of goal selection and selection of adequate test data.

c) Path Oriented test data generation:

In path-oriented test data generation the typical approach is generation of a control-flow graph. In this approach, at first a graph is generated first and subsequently, by using the graph a particular path is selected. With the help of a technique such as symbolic evaluation (in the static case otherwise it is called function minimization) test data is generated for that path in the end. In symbolic execution variables are used instead of actual values while traversing the path.

The path-oriented approach might face the problems when generating paths/graphs, traversing test data through branches and predicates (infeasible path problem), and while complexity of data types. It is harder to find the test data.

d) Heuristic technique:

It is a part of random testing but the difference with random testing is that in random testing the input test data are generated without any information but in heuristic technique some information is available prior to the testing process.

In our research project, we are trying to implement this heuristic technique, since it can provide us the most efficient way for generating the test data.

TESTING FUNDAMENTALS

2.1 What is Software Testing?

Testing a program consists of subjecting the program to a set of test inputs or test cases and observing if the program behaves as expected. If the program fails to behave as expected then the condition under which the failure occurs are noted for later debugging or correction. The general goal of software testing is to affirm the quality of a program through systematic exercising of the code in a carefully controlled environment.

2.2 Why software testing is needed?

Software testing is needed for the following reasons:

a) To provide confidence in the system
b) To identify the area of weakness
c) To establish the degree of quality
d) To provide an understanding of the overall system
e) To provide the system both useful and operable.

2.3 Types of Testing:

In general testing are of two types. They are –

a) Black – Box Testing
b) White – Box Testing

2.3.1 Black – Box Testing –

In black – box testing test cases are designed from an examination of the input / output values and no knowledge of design or code is required. The following are the two main approaches to design black – box cases:

a) Equivalence Class Partitioning: In this approach, the domain of input values to a program is partitioned into a set of equivalence classes. The partitioning is done such that the behaviour of the program is similar to every input data belonging to the same equivalence class.

b) Boundary value analysis: A type of programming error frequently occurs at the boundaries of different equivalence classes of inputs. Programmers often fail to see the special processing required by the input values that lie at the boundary of different classes. For example, programmers may improperly use < instead of <=. Boundary value analysis may lead to selection of test cases at the boundaries of different equivalence classes.

2.3.2 White – box testing

It uses the source code of the program. There are several white – box strategies and each strategy is based upon some heuristics. One – white box testing is said to be stronger than another strategy, if all types of errors detected by the first testing strategy (say A) are also detected by the second testing strategy (say B) and the second strategy additionally detects some more types of errors.

2.4 Phases of Testing:

Again testing can be divided into three phases:

a) Coding and Unit Testing
b) Integration Testing
c) System Testing

2.4.1 Coding and unit testing:

The purpose of this phase of software development phase is to translate the software design into source code. Each component of the design is implemented as a program module. The end product of this phase is set of program modules that have been individually tested. During this phase each module is unit tested to determine the correct working of all the individual modules.

2.4.2 Integration testing:

Integration of different modules is undertaken once they have been coded and unit tested. During this phase the modules are integrated in a planned manner. During each integration step, the partially integrated system is tested and a set of previously planned modules are added to it. Finally, when all the modules have been successfully integrated and tested, the system testing is carries out.

2.4.3 System Testing:

System Testing usually consists of three different kinds of testing activities:

a) Alpha Testing : It is the system testing performed by the development team
b) Beta Testing: It is the system testing performed by a friendly set of customers.
c) Acceptance Testing: It is the system testing performed by the customer himself after the product delivery to determine whether to accept or reject the delivery of the product.

The Method

In my approach, I have employed Heuristic Method of test data generation to generate the test data for particular software under test. Heuristic method is different from the other techniques as it uses some prior information that is available to generate the test data. In The total weight calculated for each set of test data will provide us with the required test data which will give optimality in path traversal.

The different steps are as follows:

- Take one sample program first.
- Then map the control flow graph of the sample program.
- Next, assign weights to the edges of the control flow graph.

- Then design a few test cases solving the individual constraints and calculate the total weights for each of the test cases. We will also find out the different path traversed by each test data.
- Based on the results, analyze the best test case data which will optimize path traversal.

For programs having loop structure:

Here, the weights have been assigned in the following manner

For sequential statements, the same weight has been assigned for the current edge as that of the previous one.

If there is a loop from a node, then the maximum weight is given to that edge and the minimum to the edge having no loop(like 80% and 20%)

If there is a conditional statement, then weight is assigned as 50%-50% of the previous edge on the current edges.

For programs having non – loop structure :

Here, we have assigned the weights in the following manner

If n is the weight assigned to a node, then the edges from that node will be having the weights as (n/2) +2 and (n/2) -2 [in case of a node having two edges].

RESULT

Result of Sample Program - 1

Control flow graph

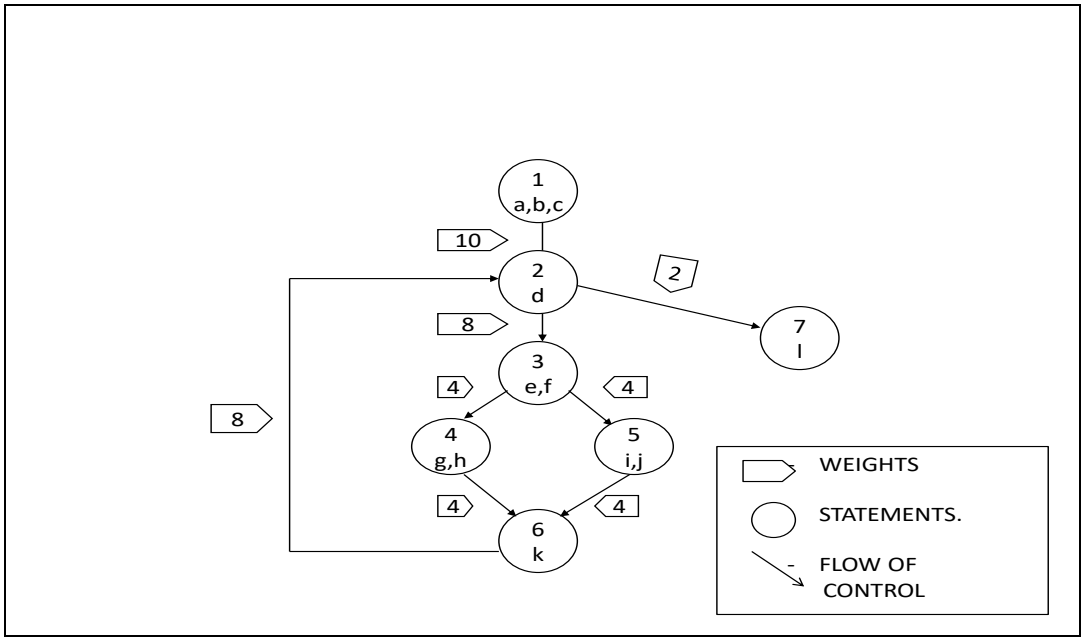

Fig.CFG of the sample GCD program

Path Table

X	Y	PATH COVERED	TOTAL WEIGHT
2	**4**	**a, b, c, d, e, f, i, j, k, d, l**	**36**
1	**1**	**a, b, c, d, l**	**12**
5	**3**	**a, b, c, d, e, f, g, h, k, d, e, f, i, j, k, d, e, f, g, h, k, d, l**	**84**

Path table of Sample GCD program

Equivalence class of the Path Table

X	Y	PATH COVERED	TOTAL WEIGHT
5	**3**	**a, b, c, d, e, f, g, h, k, d, e, f, i, j, k, d, e, f, g, h, k, d, l**	**84**

Equivalence Class of the Path table of Sample GCD program

Result of Sample Program - 2
Control flow graph

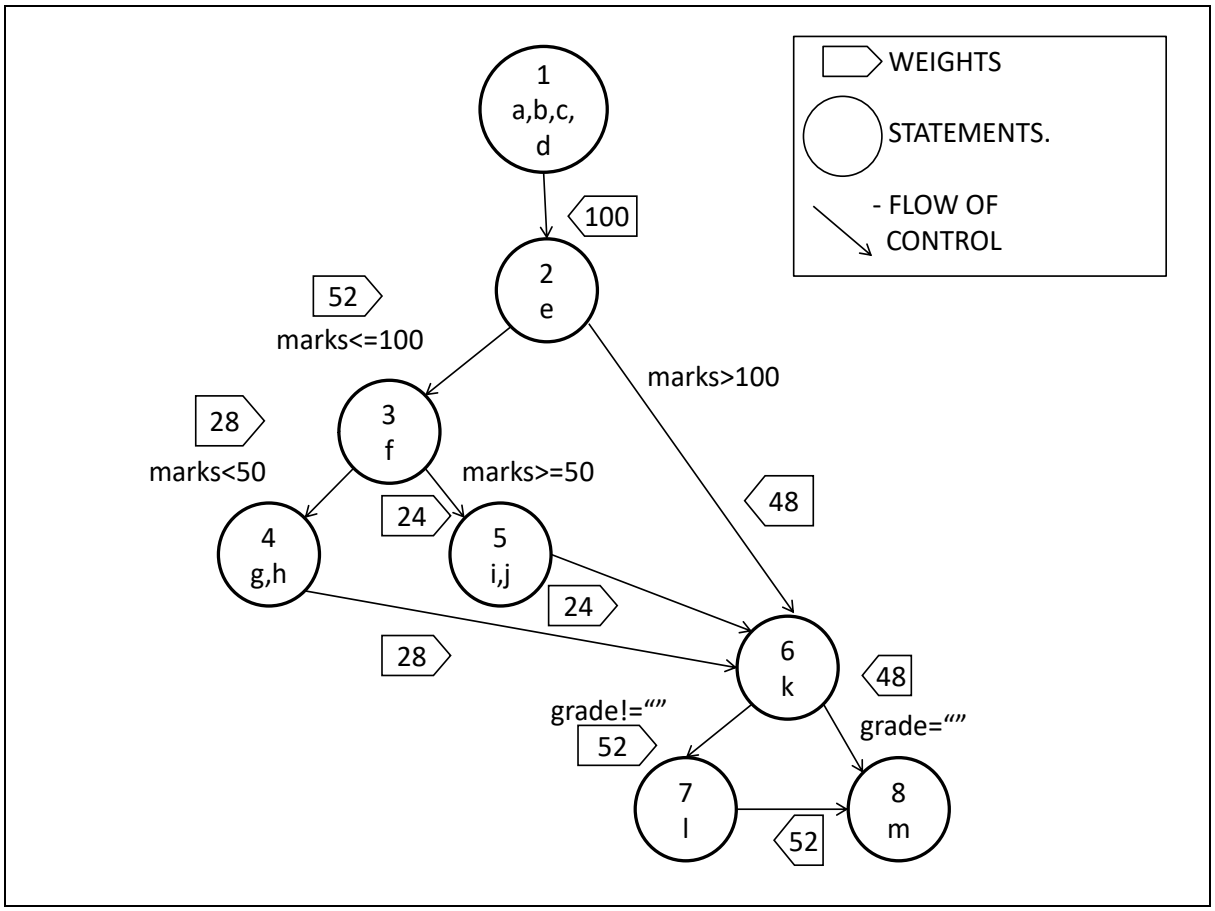

Fig. CFG of the sample Grade program

6.2.2 Path table

MARKS	PATH COVERED	TOTAL WEIGHT
99	**a,b,c,d,e,f,i,j,k,l,m**	**304**
101	**a,b,c,d,e,k,m**	**196**
49	**a,b,c,d,e,f,g,h,k,l,m**	**312**
51	**a,b,c,d,e,f,i,j,k,l,m**	**304**

Path table of Sample Grade program

Equivalence class of the Path Table

MARKS	PATH COVERED	TOTAL WEIGHT
99	**a,b,c,d,e,f,i,j,k,l,m**	**304**
101	**a,b,c,d,e,k,m**	**196**
49	**a,b,c,d,e,f,g,h,k,l,m**	**312**

Equivalence Class of the Path table of Sample GCD program

CONCLUSIONS

The proposed method generated almost all the linearly independent feasible paths. As for example, consider the GCD program. There are three linearly independent paths. And in this approach, the test data for all the paths have been generated. For the sample program "grade", we have got three test data, though there are six independent paths in the CFG.Thus the proposed method generates almost all the feasible paths that exist in a program and the presence of infeasible paths does not create any problem.

REFERENCES

[1] Arnauld Gotlieb and Bernard Botella , "Test Data Generation Using Heuristic Technique".

[2] Jon Edvardsson Department of Computer and Information Science, Linkoping University, Sweden "A Survey on automatic test data generation".

[3] Minh Ngoc Ngo and Hee Beng Kuan Tan , "Heuristic – based infeasible path detection for dynamic test data generation".

[4] Bogdan Korel, Member IEEE, "Automated Software Test Data Generation".

[5] P David Coward "Symbolic Execution And Testing ".