Automated Test Data Generation via Path Table

Ashif Ali
ashifali76@gmail.com

Jana Shafi
janashafi09@gmail.com

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.

Today testing is done manually by most of the industry due to high cost and complexity of automation. Automated testing can reduce the cost of software significantly. Automated Software Test Data Generation is an activity that in the course of software testing automatically generates test data for the software under test. Most of the automated test data generation uses constraint solver to generate test data. But it cannot generate test data when the constraints are not solvable. Although method can be found to generate test data even if the constraints are unsolvable, but it is poor in terms of code coverage. Automated testing is a good way, to cut down time and cost of software development. It is seen that for large software projects 40% to 70% of development time is spent on testing. Therefore automation is very much necessary. Test automation is a process of writing computer programs that can generate test cases, or else test cases need to be generated manually. Automated testing save time, money. In this paper, we propose a test data generation method as our aim is to improve test coverage and to avoid the unsolvable constraint problem. Our method is based on the individual constraints, and same or dependent variable to create the path table which holds the information about the path traversed by various input test data. We have implemented this method and shown along with also sample program. The result is generated in the form of table of test cases and cause – effect graph.

Keywords — Independent feasible path, Scalability, Equivalence class.

I. INTRODUCTION

Generally software is tested for its structure or functions which are supposed to be performed by its components. Testing in the prior case is commonly known as structural or white box testing, while in the later case it is called functional or black box testing [5]. For either of the cases test data is required to “traverse” through the SUT. The outcome of this traversal determines correctness, performance, or in general, the quality of the Software under test (SUT). In some cases the test data is already available or given but in most cases it is required to be generated. Test data generation is a complex activity because it involves so many steps and each step has several related issues. The architecture of a test data generator can better elaborate this notion of complexity. Fig.2 (initially drawn by [6]) shows the architecture of an automated test data generator using the path-oriented approach, in simple and easily understandable format. Fig.3 shows an overview of automated Test data Generation Techniques.

1.1 Functional Testing Techniques
These techniques are based on the functionality of the software, where test cases are derived from the program’s
specification. Examples of such techniques are equivalences partitioning, boundary value analysis, random testing and functional analysis-based testing.

1.2 Path-oriented Approach of TDG

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 [7], [8]. 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. Developing software to test the software is called as test automation/automated testing. In simple terms, automated testing is automating the manual testing process. It is used to replace or supplement manual testing with a suite of testing tools. Automated testing tools [3] assist software testers to evaluate the quality of the software by automating the mechanical aspects of the software testing task. Automated testing tools vary in their underlying approach, quality and ease of use.

Automated testing is a good way to cut down time and cost of software development. It is seen that for large software projects 40% to 70% of development time is spent on testing. Therefore automation is very much necessary. Test automation is a process of writing computer programs that can generate test cases, or else test cases need to be generated manually. Automated testing saves time, money and increase test coverage. Software testing tools and technique usually lack in portability, reliability, applicability, and scalability. Many test methods are based on the program path and our approach to detect infeasible paths using actual value execution.

1.3 Survey of related works

The literature of test data generation methods says that the main problem of symbolic execution [10],[12] is constraint solving [13]. Either constraint solving may take more time or some constraints are not solvable. Test case prioritization is used to improve the rate of fault detection especially for regression testing [5], [14], [15]. In [16], constraint prioritization technique with sampling scores method is used to deal with problem of constraint solving. The steps of the Xio methods are construction of control flow graph, finding edge priority, finding complementary pairs, and sample table and sample scores. But this method [16] works only for specific programming constructs. In this paper, in section A we discuss the approach towards the automatic generation of test cases and in section B we consider an experiment by using a problem.

II. SECTION A

2.1 Steps of Our Approach

To improve the coverage of the sample program which is the major drawback in the method as suggested by [16], we propose a method based on the variables involved in the constraint. Fig.1 shows the flow graph of our method. Fig.4 shows the steps of this method.

2.2 Source Program

A programming language is specially designed to facilitate the work of computer programmers, who specify the actions to be performed by a computer mostly by writing source code, which, for some languages, then is automatically, translated to binary machine code [1], [15], [4] that the computer can directly read and execute. An interpreter translates to machine code and executes it on the fly, while a compiler only translates to machine code that it stores as executable files; these can then be executed as a separate step.

2.2.1 Constraint Collector

The constraints in the program are found out with the help of this tool and now in another words we can say that it is flow of control.

2.2.2 Dependent variable Collector

The variables which are either same or dependent on each other are found out with the help of this tool.

2.2.3 Variable Equivalence Class Generator

Initially the equivalence class is blank. The first constraint is added to the equivalence class. Next the second constraint is checked with the first constraint and if the variable in the second constraint is either same or dependent on the variable of the first constraint then the second constraint is added to the same class or else the second constraint is added to a new class. Similarly for all the constraints the variables are checked with the variables of the constraint already in the equivalence class, to find out if they are either equal or dependent. Then they are added to the same class on which the dependent/same
2.2.4 Input Data Generator

The input test data is generated on the basis of the equivalence class of variables.

3. Path Table Flow Graph

Fig. 5 shows the path table flow graph which cases generated.

4. Path Table Algorithm

1. Path = NULL
2. Current node = start
3. If Current node != End
4. go to step 6
5. else go to step 13
6. Path = Path + Current node
7. if Current node has only one child node then
8. Current node = Child node
9. Otherwise, if constraint at the node is true then
10. Current node = Left Child node
11. Else Current node = Right child node
12. go to step 3
13. Path = Path + End

III. SECTION B

5. Experiment

5.1 We are conducting an experiment by using an problem to compute previous date, given date, as input and perform Decision table testing.

```c
#include<iostream.h>
#include<conio.h>
#include<math.h>

void main()
{
    clrscr();
    int d, m, y;
    int flag=1;
    cout<<"RULES ::
1.Range of Year::
1900-2025
2.Date should be in numeral form"
    cout<<"Enter any date (dd mm yyyy):";
    cin>>d>>m>>y;
    if(y<1900 || y>2025) flag=0;
    else
    if((y%4=0)\((y%400==0))
    { 
        If (m=2)
        (if(d<=29&&d>1)
        d--;
        else
        if(d==1)
        {
            m--; d=31;
        }else flag=0;
        else
        if(d=1)
        {if(m>=1 && m<=7)
        if(m%2==0 && m!=2)
        
        [m--; d=31;
        }else if(m%2!=0)
        
        [if(m=1)
        [y--; m=12; d=31;
        ]else
        [m--; d=30;]
        }
        }else
        if(m<=12 && m>-8)
        (if(m%2==0)
        [m--; d=30;
        ]else
        if(m%2!=0)
        [m--; d=31;]
        }
        }else
        if(m==2)
        {if(d<=28 && d>1)
        d--;
```
6. Decision Table Test

<table>
<thead>
<tr>
<th>Cause</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1$</td>
<td>$a \ 0 \ 1 \ x$</td>
</tr>
<tr>
<td>$C_2$</td>
<td>$b \ 0 \ x \ l$</td>
</tr>
</tbody>
</table>

7. Experiment Result

Table I Cause and Effect Graph

<table>
<thead>
<tr>
<th>Cause</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_1$</td>
<td>Valid $\sqrt{\checkmark}$ $\sqrt{\checkmark}$</td>
</tr>
<tr>
<td>$E_2$</td>
<td>Not Valid $\sqrt{\checkmark}$</td>
</tr>
</tbody>
</table>

8. Test Cases Generate

8.1 Advantages of Automated Testing:

The benefits of automation include increased software quality, improved time to time market, repeatable test procedures and reduced testing costs [4]. We shall now list
some more benefits of test automation. They are given below:
1. Automated execution of test cases is faster than manual execution. This saves time.
2. Test automation can free test engineers from mundane tasks and make them focus on more creative tasks.
3. Automated tests can be more reliable and immediate testing as it need not wait for the availability of test engineers. In fact

   Automation = Lesser Person Dependence

4. Tests, once automated, take comparatively far less resources to execute. A manual test suite requiring 10 persons to execute if over 31 days, i.e., 31*10=310 man days, may take just 10 man-days for execution, if automated. Thus a ratio of 1:30 is achieved.
5. Automation does not end with developing programs for the test cases. Automation includes many other activities like selecting the right product build, generating the right test data, analyzing results and so on.

8.2 Disadvantages of Automated Testing:
Despite of many benefits, pace of test-automation is slow. Some of its disadvantages are given below [4]:
1. An average automated test suite development is normally 3-5 times the cost of a complete manual test cycle.
2. Automation is too cumbersome. Who would automate? Who would train? Who would maintain? This complicates the matter.
3. Automation is not an item of higher priority for managements. It does not make much difference too many organizations.

IV. DISCUSSION

Today the cumbersome task of software testing is relying more and more on the support received from automated tools. The automation of software test data generation is of particular concern to the researchers. To conclude the discussion it can be said that a big gap exists in research on some of the ATDG areas like intelligent approach, specification-based approach, and component-based approach. Additionally, the discussion highlighted some trends like ‘use of meta heuristic and intelligent approach’, in the research which did not get the due importance, but it could prove worthy for ATDG if it is adopted on a large scale. Likewise taxonomies of ATDG techniques could highlight certain aspects carrying worth for further research and produce new techniques as a result of that. It was almost impossible for the author to discuss each and every aspect of the work therefore only the hot moot points have been highlighted in the discussion section. Valid conclusions from the ensuing discussions and on-spot recommendations have also been included in the research work.

A. Application of Automated Testing

1. The performance of computer hardware has been improved rapidly.
2. Make improvements to the search algorithm.

3. The constraint solving approach is better in terms of efficiency and degree of automation.
4. A large part of current testing research aims at improving the degree of attainable automation[1], either by developing advanced techniques for generating the test inputs (this challenge is expanded below), or, beyond test generation, by finding innovative support procedures to automate the testing process.

V. COMPARISON OF AUTOMATED TESTING WITH VARIOUS TECHNIQUES

Fig.6 compares the automated testing with various other existing methods and below discuss briefly also

\[\text{Fig.6 ATDG Techniques}\]

1. Functional Testing Techniques: These techniques are based on the functionality of the software, where test cases are derived from the program’s specification. Examples of such techniques are equivalence partitioning, boundary value analysis, random testing and functional analysis-based testing [17].
2. Structural Testing techniques: Structural ATDG techniques are based on the internal structure of a program, where test cases are selected so that structural components are covered (i.e. all paths are exercised in the SUT). Examples of such techniques are statement testing, branch testing, path testing, predicate testing, dataflow testing, structured testing, and domain testing [17]. Further structural techniques could be based on different approaches and implementation methods.
3. Random Approach of TDG: According to [18] 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.
4. Path-oriented Approach of TDG: 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 [19, 20]. 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.

5. Goal-oriented Approach of TDG: 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 [19]. 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 very selected statement (respective branch) [23]. Two typical approaches, ‘Assertion-based’ and Chaining approach are known as goal-oriented. In the first case assertions are inserted and then solved while in the second case data dependence analysis is carried out. Generally the goal-oriented approach faces issues of goal selection and selection of adequate test data.

6. Intelligent Approach of TDG: According to Pargas et.al [19] the intelligent test-data generation approach often relies on sophisticated analysis of the code to guide the search for new test data. However in the author’s opinion this approach can be extended up to the intelligent analysis of program specification as well. With the proposed extended ability this approach will fall in between functional and structural testing. At this time this approach is quite limited in use (just reported by [19, 24, and 25]) therefore, its specialization in use or its pros and cons cannot be stated with any certainty.

7. Static Methods: These are the testing methods adopted for analysis and checking of system representations such as the requirements documents, design diagrams and the software source code, either manually or automatically, without actually executing the software [22],[26]. In other words, the static methods do not require the software under test to be executed. They generally use symbolic execution to obtain constraints on input variables for the particular test criterion. Solutions to these constraints represent the test-data [25]. Executing a program symbolically means that instead of using actual values, ‘variable substitution’ is used.

8. Dynamic Methods: Instead of using variable substitution, these methods execute the SUT with some, possibly randomly selected input [22]. By monitoring the program flow the system can determine whether the intended path was taken or not. In case of a negative answer the system backtracks to the node where the flow took the wrong direction. Using different kinds of search methods the flow can then be altered by manipulating the input in a way so that the intended branch is taken.

9. Hybrid Methods: As the name suggests, a hybrid method is the mixture of static and dynamic methods. This mixture could be in the form of side-by-side use, probably solving some tasks by using static methods and some by using the dynamic methods, or it could be a synthesis of static and dynamic methods. Such an ATDG technique is reported by [21] where the symbolic execution is used for a dynamic approach.

VI. FUTURE SCOPE

In future we will further research for finding methods for feasible paths of our program. The method should be tested with more examples for accuracy. The method should improve the scalability by including more data types like dynamic data structure and string. Hopefully, future work either in the form of survey of some specific area or a systematic review, would be based on the knowledge produced in this systematic review. Computing trends are diverting more and more towards the independent development of components. It is, therefore, recommended that this trend should also be considered for the research on ATDG techniques.

VII. CONCLUSION

The proposed method has generated test data almost for all the feasible independent paths. As for example for the sample program1, t Coverage is inversely proportional to the number of infeasible paths. The main issue of test data generation using this method is how to take input for finding the paths. We tested our input test data algorithm with program. Our experimental results show that this method can generate reliable test. Data at a lower cost but it is not optimum. We describe an approach for test data generation with lower computation cost. The main issue of test data generation using this method is how to take input for finding the paths.

ACKNOWLEDGMENT

Authors want to thanks the almighty Allah has provided the ability, the resources, the opportunity, and kind support. Every word of thanks and praise ultimately refers to Alahand special thanks to our parents, brothers and sisters who significantly contributed in upbringing and brought to a stage where we can make humble contributions to human knowledge.

REFERENCES

and Engineering in Linkoping, pages 21-28, ECSEL, October 1999

generation using genetic algorithms. Software Testing,

[8] AkosHajnal and IstvanFogaras. An Applicable Test Data
Generation Algorithm for Domain Errors. In proceedings of the
1998 ACM SIGSOFT international symposium on Software
testing and analysis ISTAS98; Vol. 23, No. 2, 63-72/98, ISSTA 1998.

[9] ShahidMahmoud,”A Systematic Review of Automated Test
Data Generation Techniques”, School of Engineering, Blekinge
Institute of Technology Box 520 SE-372 21 Ronneby, Sweden,
October 2007.

Pointer and Scture Variables” Proceedings of the Fourth
International Conference on Quality Software(QSIC’04) IEEE

test data generation using symbolic execution and constraint solving
techniques”. In Proceedings of the Interna- tional Conference on

[12] Lori A. Clarke. , “A system to generate test data and
symbolically execute programs.” IEEE Trans. On Software


[14] Chengyunn Chu Mary Jean Harrold Gregg Rother- mel, Roland
H. Untech, “Test case prioritization: An empirical study,” In
Proceedings of the International Conference on Software

“All code coverage is not created equal” A case study in
prioritized code coverage.” Technical report, IBM Toronto Lab,
2006.

[16] J. Jenny Li. “Prioritize code for testing to improve code coverage
of complex software.”, 2005.


[18] BogdanKorel. Automated Test Data Generation for Programs
with Procedures.Inproceedings of the 1996 ACM SIGSOFT
international symposium on Software testing and analysis ’96,

generation using genetic algorithms. Software Testing,

Generation Algorithm forDomain Errors.In proceedings of the
1998 ACM SIGSOFT international symposium on Software
testing and analysis ISTAS98; Vol. 23, No. 2, 63-72/98, ISSTA 1998.

using constraint logicprogramming and symbolic execution.
Software Testing, Verification and Reliability 11:81–96 (DOI:

[22] Huey-Der Chu, John E. Dobson, and I-Chiang Lui. FAST: a
framework for automatingstatistics-based testing. Software

Data Generation usingConstraint Solving Techniques. In
proceedings of the 1998 ACM SIGSOFT international symposium

Test Data Generation ForComplex Programs.In proceedings of
13th International Conference on Automated

Flaw Finding usingSimulated Annealing.In proceedings of the
1998 ACM SIGSOFT international symposiumon Software

[26] L. Parsonsseville. Software Engineering, 5th edn (Addison-
Wesley, Wokingham, 1996).

### AUTHOR’S PROFILE

**Place:** New Delhi  
**Date of Birth:** 6th Feb 1984  
**Degree:** B.Tech. (Al-Falah School of Engineering & Technology, Faridabad, Haryana, India, 2008). M.Tech. (Al-Falah School of Engineering & Technology, Faridabad, Haryana, India, 2012).  
**Major Field of Study:** Software Testing.  

His work experience includes 3.7 years as an ASSISTANT PROFESSOR and HOD, (IT Department, Brown Hills College of Engineering & Technology, Dhauj, and FBD). Presently, he is the HOD/DPC in CSE Department, Brown Hills College of Engineering & Technology, Dhauj, and FBD.

**Details of Publications:**  
1. Paper Published in InterNational Journal Conferences (Online) - 3  

2. Paper Published in Journal Conference - 1  

3. Paper Published in International Conference - 1  

Mr. Ashif Ali is awarded with the certificate of excellence for the A comparative study on Types of Functional Testing”, International Journal of Computer Science and Management Research, vol. 1 issue 2, September 2012, Pages 150-155 and ISSN: 2278-733X.And also for yielding best academic results in various subjects. He worked for plantation committee (Earth Science Day-2009). Further 1 paper is to be published and 1 paper is submitted in international journal. And 2 books are in processing for publication.

**Place:** New Delhi  
**Date of Birth:** 30th November 1989.  
**Degree:** B.Tech (Al-Falah School of Engineering & Technology, Faridabad, Haryana, India, 2010). M.Tech(Al-Falah School of Engineering & Technology, Faridabad, Haryana, India, 2012).  
**Major Field of Study:** Software Testing.  

Her work experience includes 2.2 years as an ASSISTANT PROFESSOR in CSE Department, Brown Hills College of Engineering & Technology, Dhauj, and FBD.

**Details of Publications:**  
1. Paper Published in InterNational Journal Conferences (Online) - 3  

Ms. Jana Shafi is an Assistant Professor in CSE department having qualifications as B.Tech (CSE) & M.Tech. (CE) AND awarded with the certificate of excellence for the A comparative study on Types of Functional Testing”, International Journal of Computer Science and Management Research, vol. 1 issue 2, September 2012, Pages 150-155 and ISSN: 2278-733X.And also for yielding best academic results in various subjects. She is an active member of Examination Committee. Further 1 paper is to be published and 1 paper is submitted in international journal. And 2 books are in processing for publication.

Copyright © 2012 IJAIM right reserved