International Journal of Computer Applications (0975 – 8887)
9th International ICST Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness (QShine-2013)

A Simple Technique to Find Diverse Test Cases

Shilpi Singh
Intern, Testing Vertical, CDAC Noida, R&D Department,
A Scientific Society of DEIT, Govt. of India

Chetna Sharma
Intern, Testing Vertical, CDAC Noida, R&D Department,
A Scientific Society of DEIT, Govt. of India

Udai Singh
Technical Officer,
Testing Vertical, CDAC Noida, R&D Department,
A Scientific Society of DEIT, Govt. of India

ABSTRACT
As the software modified, new test cases are added to the test suits, the test suite grows and the cost of regression testing increases. This paper defines a technique to solve these problems and make the testing cost effective. We introduce a set of test case comparison metrics algorithms which will quantitatively calculate the diversity between any arbitrary test case pair of an existing test suite. Our procedure mainly focuses on branch coverage criteria, control flow of a program, variable definition-usage and data values. By using these information's a signature values is calculated to find how much the test cases are diverse from each other and accordingly we can make a cluster of similar test cases together, that can effectively test under time constraints.

Keywords
Testing, Program Testing, Regression Testing, Test Suite, Test Case, Test Suite Reduction, FDE.

1. INTRODUCTION
Program testing is the most commonly used method for demonstrating that a program accomplishes its intended purpose. It is an opportunity to deliver quality software and to substantially reduce development cost as much as possible. It also involves selecting elements from the program's input domain, executing the program on these test cases, and comparing the actual output with the expected output [1][5].

Test suite, for testing and validating software program can contain large number of test cases to execute various part of the software program code. The main aim is to check for defect and code compliance. The general size of a test suite can vary from hundreds of test cases to more than a million for large and/or evolved software program. Thus, test execution can take a great deal of time to complete. Additionally, test cases are often developed and thus one or more test cases in a test suite may be redundant. In that they execute the same code path for the same or similar data sets. Moreover, as the software program is updated and modified, test cases in a test suite can become duplicative. Thus, it would be advantageous to identify and eliminate redundant test cases in a test suite, allowing reduction in the test suite size which, in turn, can decrease testing time and contribute to optimize maintenance effort for the test suite. Further, it would be advantageous to perform test suite clustering of test cases based on a defined similarity level. Using clustering, the number of test cases to be executed for any particular test run can be limited and even minimized. Test case clustering support an identification of a minimal set of test cases that maximizes test coverage of the software program while minimizing the number of tests to be run.

2. TEST SUITE REDUCTION
Regression testing is an important activity to the development and maintenance of evolving software. Difficulty is that, it requires large amount of test cases to test new or modified parts of the software. Test-suite reduction techniques attempt to reduce the costs of saving and reusing test cases during software maintenance by eliminating redundant test cases from test suites. According to Rothermal [3], a Software contains 10 to 20,000 of LOC (Lines of Codes) requires more time and effort to run all the test cases. To eliminate duplicate or redundant test cases in a test suite and to optimize the testing process, test suite minimization approach is necessary. The first formal definition of test suite reduction problem introduced in 1993 by Harrold et al. [2] as follows:
Given: $\{t_1, t_2,\dots, t_m\}$ is test suite T from m test cases and $\{r_1, r_2,\dots, r_n\}$ is set of test requirement that must be covered in order to provide desirable coverage of the program entities and each subsets $\{T_1, T_2,\dots, T_n\}$ from T are related to one of r_is such that each test case t_j belonging to T_i satisfies r_i.
Problem: Find minimal test suite T' from T which satisfies all r_is covered by original suite T.
The main aim of test suite reduction is to optimize the testing process while maintaining Fault Detection Effectiveness (FDE) [4].

3. PROPOSED TEST CASE COMPARISON METRICS ALGORITHM
This section introduced four metrics to compare any two test cases of a test suit in a quantifiable manner. It is also used for analyzing, comparing & clustering. Quantitative test case comparison metrics suggest the amount of similarity between any test case pair, that is being tested on target code. Key aspects of program execution including code coverage, counts of execution, data values and def-use of variables are captured. Program code is divided into parts depending on the metric (Ex. Control divergence has control statements). Signature values are calculated for each part based on test execution and these values are used for comparing pairs of tests.

3.1 Metrics
This section explains the four metrics through which distance between any two test cases is being calculated i.e. how much they are different or diverse from each other. Following are the four metrics:
1. Block coverage equivalence M_1
2. Control flow divergence M_2
3. DU equivalence M_3
4. Data divergence M_4

3.1.1 Block Coverage Equivalence Algorithm
Block coverage equivalence measures block testing overlap between two test cases of a test suit. M_1 metric is calculated by

adding the number of common blocks tested by a test case pair and dividing this sum by the total number of unique blocks tested by both test case pair. In this metric there is no requirement that the test cases of the pair execute the common blocks in the same order or the same time frame. See Fig. 1.

```
Given: N = Total no. of test cases
X = No. of common blocks between $T_i$& $T_{i+1}$
Y = No. of unique blocks tested by both $T_i$& $T_{i+1}$

    1. For each test case   $T_i \le N$
    2. For each test case   $T_{i+1} \le N$
    3. $M_1(T_i,T_{i+1}) = X / Y$
    4. End for
    5. End for
```

Fig. 1: Block coverage equivalence algorithm

3.1.2 Control Flow Divergence Algorithm

Control flow divergence measures the similarity of two test cases that test the same blocks that have conditional path within them. For each test case that executes a block with a conditional branch a control flow (CF) value is calculated that provides a measurement for the number of times the test case takes each conditional branch (True and False branch). See Fig. 2.

```
Control Flow (CF) is calculated for each test case that
executes block B with a conditional branch.

Given: T= No. of times true branch is executed
F= No. of times false branch is executed
    1. For each test case $T_i \le N$
//Calculate sum of the no. of times each branch is executed,
    2. SUM ( T &F) = T+F
//Calculate Average no. of times each path is executed,
    3. AVG (T&F) = SUM (T&F) / No. of Branch’s
//Find the CF value of each block corresponding to test cases,
    4. $CF_B(T_i)$ = ((T-AVG (T&F)) + (AVG (T&F)-F)) /
       SUM(T&F)
    5. End For
Variance of CF values is calculated for each test case pair
that executes common blocks with conditional statements

Given: N= No. of shared block of test case pair $T_i$&$T_{i+1}$
Calculate, M = MEAN ($CF_B(T_i)$ , $CF_B(T_{i+1})$)
    1. For each test case   $T_i \le N$
    2. For each test case   $T_{i+1} \le N$
//Calculate the variance of CF value for a common block B of
$T_i$& $T_{i+1}$
    3. Variance $\Delta B(T_i,T_{i+1})$ = Square[$CF_B(T_i)$-M] +
       Square[$CF_B(T_{i+1})$-M]
//Calculate metric value for each test case pair
    4. $M_2(T_i,T_{i+1})$ = Sum of the variance of common
       blocks($\sum\Delta B(T_i,T_{i+1})$ ) / No. of common blocks
    5. End For
    6. End For
```

Fig. 2: Control Flow Divergence Algorithm

3.1.3 DU Equivalence Algorithm

Our path selection criteria are based on an investigation of the ways in which values are associated with variables, and how these associations can affect the execution of the program. This analysis focuses on the occurrences of variables within the program. Each variable occurrence is classified as being a definitional, computation-use, or predicate-use occurrence. We refer to these as def, c-use, and p-use, respectively [1].

DU equivalence measures def-use (definition or use) path testing overlap between two test cases in a test suit. A def-use path is a logic execution sequence in a block that defines and uses a variable. The du-paths and dc-paths describe the flow of data across source statements from points at which the values are defined to points at which the values are used. The du-paths that are not definition clear are potential trouble spots. See Fig. 3.

```
Given: N= No. of test cases
X= No. of common def-use chain tested by test case
pairs $T_i$& $T_{i+1}$
Y= No. of unique def-use chain tested by both test
cases pairs
    1. For each test cases $T_i \le N$
    2. For each test cases $T_{i+1} \le N$
    3. $M_3(T_i$& $T_{i+1}) = X/Y$
    4. End for
    5. End for
```

Fig. 3: DU Equivalence Algorithm

3.1.4 Data Divergence Algorithm

Data divergence measures the similarity or diversity of test cases with respect to the data values the test cases use for code variables. Data divergence is computed as the similarity of the number of times two test cases execute the same conditional branches, loops and /or blocks. Counts are used to indirectly represent the data values of variables in the software code under test. See Fig. 4.

```
For each test case that executes a block B a data flow DF
value is calculated
Given: N= total no. of test cases
       T=No. of times true branch is executed
       F=No. of times false branch is executed
    1. For each test case $T_i \le N$
//Calculate data flow DF($T_i$ , B)
    2. DF($T_i$ , B) = (T-F)/2
    3. End For

For each test case pair calculate the variance of DF value for
a common block B with loop statements executed by $T_i$&$T_i$+1

    1. For each test case $T_i \le N$
    2. For each test case $T_{i+1} \le N$
    3. M=Mean ($DF_B(T_i)$), ($DF_B(T_{i+1})$)
    4. Variance $\Delta B(T_i,T_{i+1})$=Square ($DF_B(T_i)$ - M)+
       Square($DF_B(T_{i+1})$ - M)
//Calculate metric value is for each test case pair
    5. $M_4$ ($T_i$, $T_{i+1}$) = Sum of the variance of common
       blocks/No. of common blocks
    6. End For
    7. End For
```

Fig. 4: Data Divergence Algorithm

3.2 Calculation of Signature Values for Each Test Case Pair

This section describes how signature values are calculated for each test case pair. For each test case pair two signature values are generated. These are:

1. Equivalence Signature Value
2. Divergence Signature Value

Each test case signature is an aggregate quantifiable metric that is used to identify the amount of similarity or dissimilarity of the test cases of a test case pair. A signature is a weighted average of a subset of the K metric generated for a test case pair.
Thus, once a set of metric value are established for a test case pair each metric is weighted, a subset of the weighted metrics are summed, and the result is divided by the number of added metrics, to define signature for the test case pair. Each metric is given equal weight, or importance, and thus the weight assigned each metric is one. A metric can be disabled by assigning it a weight of zero.

Equivalence Signature = $(P_1 M_1 + P_3 M_3)/2$ (1)

Where, $P_1 = P_3 = 1$ (P_x is the weight for the x^{th} metric),
M_x is the x^{th} metric value for test case pair,
M_1= Block coverage equivalence metric value, M_4 = DU equivalence metric value.

Divergence Signature = $(P_2 M_2 + P_4 M_4)/2$ (2)

Where, $P_2 = P_4 = 1$ (P_x is the weight for the x^{th} metric),
M_x is the x^{th} metric value for test case pair,
M_2 = Control Flow Divergence metric value,
M_4 = Data Flow Divergence metric value.

Calculated signature values for each test case pair of a test suite are stored and used to group the test cases into one or more cluster.

4. EXPERIMENTAL STUDY

In this section an experiment is done on the small program (written in C++ language) which is shown in fig. 5. Here B and D represents block/branch and decision block respectively.

Table 1 shows the developed test cases inputs for the sample program mention in the fig 5 and table 2 shows the blocks or branch covered by each test case. Similarly for the control flow divergence, a matrix is created which shows number of times true and false branch is covered by all the test cases. Remaining is also created accordingly. With the help of these information's, four metrics M_1, M_2, M_3 and M_4 are calculated

```
int main()
            {
char ch;
  B1:       int num i ;
  B21:      cout<< "enter a number \n";
            cin>>num;
  D1:                 if(num%2==0)
  B22                 cout<< "even\n";
                      else
  B23                 cout<< "odd\n";
  D2                  if(num==1)
  B24                 cout<< "prime\n";
  B25                 else
                          {
            for(i=2;i≤num/2;++i)
  D3                            if(num%i==0)
                                {
  B251                          cout<< "not prime\n";
                                gotolb;
                                }
  B252                          cout<< "prime\n";
                      }
                      lb;
            RETURN 0;
}
```

Fig. 5: Sample Program

Table 1. Test Cases

Test Case	Value of N
T1	7
T2	2
T3	6
T4	15
T5	1
T6	10

Table 2. Block Coverage Matrix

Test Cases	Blocks Covered by Test Cases				
T1	B_1	B_{21}	B_{23}	B_{25}	B_{252}
T2	B_1	B_{21}	B_{22}	B_{25}	B_{252}
T3	B_1	B_{21}	B_{22}	B_{25}	B_{251}
T4	B_1	B_{21}	B_{23}	B_{25}	B_{251}
T5	B_1	B_{21}	B_{23}	B_{24}	
T6	B_1	B_{21}	B_{22}	B_{25}	B_{251}

5. EXPERIMENTAL RESULT

By applying the metrics value on equation 1 and 2 the signature values are calculated. These values represent how much the test cases are similar or distinct from each other. For this threshold values for equivalence signature and divergence signature are defined. According to these threshold values, weather two test cases are allowed to make a group (cluster) or not can be easily defined. Two test cases are deemed similar to group or cluster if the divergence signature value for the test case pair is less than or equal to a pre-defined divergence threshold and the equivalence signature value for the test case pair is greater than or equal to a pre-defined equivalence threshold.

Threshold values are chosen according to the user's need. Larger the divergence threshold value and smaller the equivalence threshold value, larger the cluster will be. Larger cluster results in less test cases that may need to be run, reducing test time and effort. Here, rigid threshold values 1 and 0 are chosen for equivalence and divergence signature to check identical test cases.

Table 3 shows the calculated signature values for each test case pair of the above program. Each cell contains signature values for corresponding test case pair. Upper and lower value represents divergence signature value and equivalence signature value respectively. For making a group or cluster of similar test cases threshold values of divergence and equivalence as 0.5 and 0.98 are assumed. Signature values of test case pair T3-T6 are 0 and 1, it means they are identical and we can remove one of them. We can also make a group of T1-T5 because it approximately satisfies the given condition as compare to others. See Figure 6.

6. CONCLUSION AND FUTURE WORK

In this paper we presented the four test case comparison metrics algorithms that are simple and easy to understand. By using these algorithms we can easily capture the diversity level in terms of signature values between two test cases. Accordingly clustering or grouping of test cases is processed which reduces the testing time as well as cost. Currently we are working on effective prioritization of test cases in a cluster. In future we would like to run similar experiments on programs from a broader range of programming languages, sizes and problem domains.

Table 3. Signature Matrix for Test Case Pair

Test Cases	T1	T2	T3	T4	T5	T6
T1		0.69 0.83	0.36 0.58	0.17 0.70	0.00 0.58	0.36 0.58
T2			1.00 0.70	1.06 0.58	0.00 0.47	1.00 0.70
T3				0.06 0.83	0.00 0.39	0.00 1.00
T4					0.00 0.50	0.06 0.71
T5						0.00 0.39
T6						---

7. ACKNOWLEDGMENTS

The authors wish to thanks Udai Singh, Technical officer, CDAC Noida for supporting this work and for his valuable suggestions. We are also grateful to an anonymous reviewer who made several very helpful suggestions and detailed comments that improved the paper.

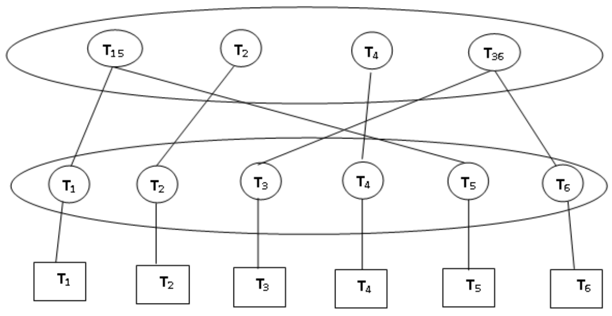

Figure 6: Clustering Process

8. REFERENCES

[1] Sandra Rapps, Elaine J. Weyuker, "Selecting software test data using data flow information", IEEE Transactions on Software Engineering", Vol.SE-1l, No. 4, April 1985.

[2] Gregg Rothermel, Mary Jean Harrold, Jeffery von Ronne, Christie Hong, "Empirical Studies of Test-Suite Reduction," Journal of Software Testing, Verification, and Reliability. 12(4):219-249, 2002.

[3] G. Rothermal, R. H. Untch, C. Chu, M. J. Harrold, "Prioritizing test cases for regression testing", IEEE Transcations on software Engineering, 27 (10)(2001) 929-948Tavel, P. 2007 Modeling and Simulation Design. AK Peters Ltd

[4] D. Jeffrey and N. Gupta, "Improving fault detection capability by selectively retaining test cases during test suite reduction", IEEE Transcations on software Engineering, 33(2):108–123, 2007.

[5] Boris Beizer,"Software Testing Technique", Second Edition International Thomson Computer 1990 ISBN 1-85032-880-3