2009 International Symposium on Computing, Communication, and Control (ISCCC 2009)
Proc .of CSIT vol.1 (2011) © (2011) IACSIT Press, Singapore

Effective Tool for Test Case Execution Time Reduction

Dr. R. P. Mahapatra[1], M. Mohan[2] and A. Kulothungan[3] +

[1, 2, 3] SRM UNIVERSITY

Abstract. This paper proposes a new technique for improving the efficiency of software testing, which is based on an attempt to reduce test cases using existing techniques as well as reducing test case execution time by using proposed technique that have to be tested for any given software. The approach utilizes the advantage of Regression Testing where fewer test cases would lessen time consumption of the testing as a whole and achieve maximum coverage. As for the test case reduction by using existing method, the technique uses simple algebraic conditions to assign fixed values to variables (Maximum, minimum and constant variables). By doing this, the variables values would be limited within a definite range, resulting in fewer numbers of possible test cases to process. As for the test case Execution time reduction by using proposed method, the technique uses simple parallelism to execute test case parallely without interfering other path in the application. By doing this, we can reduce the test cases execution time and cost of testing.

Keywords: Software testing, test case generation, test case Reduction.

1. Introduction.

1.1. Software-testing techniques

With finding errors as the primary objective of software testing, higher probability of detecting defects has become the defining quality of an effective test. Computer-based systems, which are known to offer testers with diversity of testing methods and, hence, enhance probability of detection, are therefore recommended as the most efficient tools currently available.[4][6]

A strategy for software testing integrates software test case design methods into a well-planned series of steps that result in the successful construction of software.The strategy provides a road map that describes the step to be conducted as part of testing, when these steps are planned then undertaken, and how much effort, time, and resource will be required. Therefore, any testing strategy must incorporate test planning, test case design, test case execution and resultant data collection and evaluation. [1]

Another important aspect of software testing is that the number of the test cases that have a direct effect on the cost of testing, particularly that of Regression testing [1]. When tests must be run repeatedly for every change in the program, it is advantageous to have as small a set of test cases as possible.

1) Control Flow testing: Aims to inspect the validity of selected control flow without the need for testing every possible path (as required in Structural testing). The test is preferable when the number of all available paths is so great that testing all of them become impractical. [1]

2) Independent program paths: An independent program path is any path through the program that introduces at least one new set of processing statements or a new condition. When stated in terms of a flow graph, an independent path must move along at least one edge that has not been traversed before the path is defined.

3) Cyclomatic Complexity: The Cyclomatic complexity gives a quantitative measure of the logical complexity. This value gives the number of independent paths in the basis set and an upper bound for the

+ Corresponding author.
E-mail address: (mahapatra.rp@gmail.com, mmohanit.2006@gmail.com, kulosoft@gmail.com).

number of tests to ensure that each statement is executed at least once. An independent path is any path through program that introduces at least one new set of processing statements or a new condition (i.e... new edge)[1]

Example

- Number of regions of flow graph
- Edges-nodes+2
- Predicate node+1.

Deriving test cases:

Using the design or code, draw the corresponding flow graph.

Determine the Cyclomatic complexity of the flow graph.

Determine a basis set if independent paths.

Prepare test cases that will force execution of each path in the basis test.

2. PROBLEM DESCRIPTION

2.1. Issues of interest

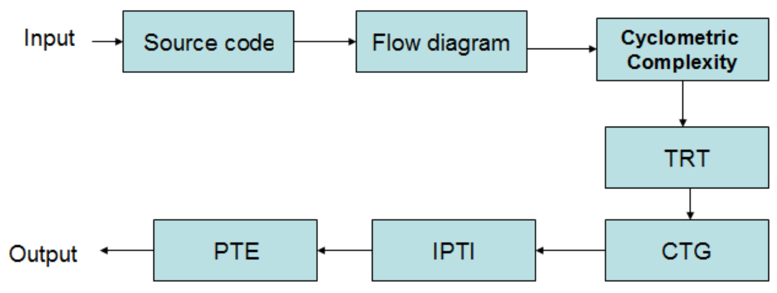
In the commercial automated testing tool, all test cases are executing sequentially. From that it takes more time to execute all test cases. So by using proposed techniques, identifying common test cases from all independent path and then executing parallely. From that we can reduce the execution time.

2.2. Interested problems

This paper tries to improve test performance as follows:

- **Automatic test case generation** – One of the most important components in a testing environment is an automatic test data generator
- **Identifying common test cases** – The technique reduces the cost of executing and validating tests. Therefore it is of great practical advantage to reduce the execution time.
- **Minimum number of test runs** – Use less time is spent on test runs.

3. OVERALL BLOCK DIAGRAM



TRT-Test case Reduction Technique

CTG-Common Test case Generation

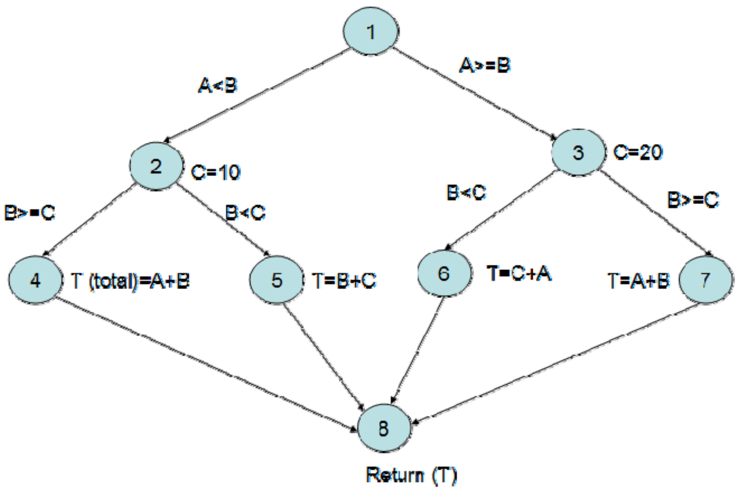
IPTI-Individual Path Test case Identifier

PTE- Parallel Test case Executor

3.1. SOURCE CODE

```
Void main () {
  Int a, b, c;
  If (a<b) {
     C=10;
     If (b>=c) {
        Total=a + b ;}
           Total=b + c ;}
       Else   {
     C=20;
     If (b<c)     {
        Total= c + a ;}
     Total=a + b;    }
  Return (total)
  }
```

3.2. FLOW DIAGRAM



3.3. CYCLOMATIC COMPLEXITY

It is used to calculate number of Independent path from flow graph.

C (g) =Number of Region+1

From the flow graph:

C (g) =3+1

Then cyclometic complexity equal to Number of independent path

Path1: 1,2,4,8

Path2: 1,2,5,8

Path3: 1,3,6,8

Path4: 1,3,7,8

3.4. TRT (TEST CASE REDUCTION TECHNIQUE)

There are four steps to generate test cases:

1) Finding all possible constraints from start to finish nodes of the control flow graph. A constraint is a pair of algebraic expressions, which dictate conditions of variables between start and finish nodes

2) Identifying the variables with maximum and minimum values in the control flow, if any. Using conditions dictated by the constraints, two variables, one with maximum value and the other with minimum value, can be identified. To reduce the test cases, the maximum variable would be set at the highest value within its range, while assigning the minimum variable at the lowest possible value of its range.

3) Finding constant values in the path, if any. When constant values can be found for any variable in the path, the values would then be assigned to the given variables at each node.

4) Using all of the above-mentioned values a table is created to present all possible test cases.

Assume that the path 1-2-4-8 is selected and the initial domains of the input variables are

<0 to 30>, <0 to 50>, <0 to 40>

The algorithm steps follow:

1) Finding all possible constraints from start to finish nodes.

A < B, B> = C

2) Find minimum values in the path, if any.

From the above conditions, it is possible to identify 'A' as the variable with the minimum value and 'B' as the variable with maximum value. In accordance to the finding, a value of zero, the lowest value within the range of variable 'B', can then be assigned to 'A' while the value of 'b' can be set at 50, the highest value of the variable.

3) Finding constant values in the path, if any. 'C' constant value found on node 2 of the path has been used to replace the fix value of C (10) at the node.

4) Using all of the above-mentioned values to create a table to present all possible test cases. 'A' value is 0...30, 'B' as the variable with maximum value = 50, 'C' as the variable with the minimum value = 10. 'B' as the value of 31...50 because of 'B' should be greater than 'A' as per constraint in the path.

Similarly, according to above technique we can calculate test case for path2, path3 and path4:

Path2:1-2-5-8

A<B, B<C, C=10 (constant)

Final Test case Range A=0...9, B=10...50, C=10

Path3:1-3-6-8

A>=B, B<C, C=20(constant)

Final Test case Range A=0...30, B=10...30, C=20

Path4: 1-3-7-8

A>=B, B>=C, C=20(constant)

Final Test case Range A=30, B=0...50, C=20

VARIABLE A	VARIABLE B	VARIABLE C	PATH
0 – 30	31 – 50	10	Path1
0 – 9	10 – 50	10	Path2
0 – 30	10 – 30	20	Path3
30	0 – 50	20	Path4

3.5. CTG (COMMON TEST CASE GENERATION)

From the table range of value for variable 'A' used in path1, path2 and path3. So Variable 'A' interval divided into 3 Range.

Formula:

Range = Total Interval / 3

From the table range of value for variable 'B' used in path1, path2, path3 and path4. So Variable 'B' interval divided into 4 Range.

Formula:

Range = Total Interval / 4

No needs to divide the variable 'C' because of all paths contain constant value.

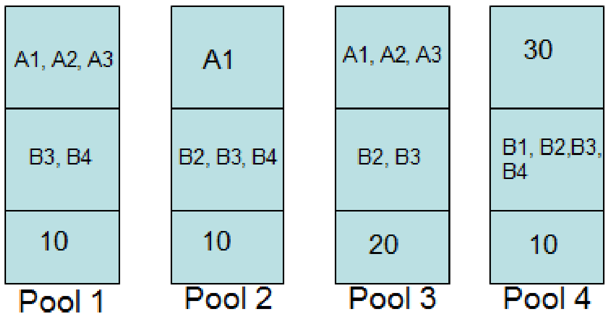
(i) Divided range of values for variable 'A' (ii) Divided range of values for variable 'B'

A1	**0-10**
A2	**11-20**
A3	**20-21**

B1	**10-20**
B2	**21-30**
B3	**31-40**
B4	**41-50**

3.6. IPTI (INDIVIDUAL PATH TEST CASE IDENTIFIER)

It generates some unique value to each interval on independent path, Here A1, A2, A3, B1, B2, B3 and B4 are range identifier in the pool.

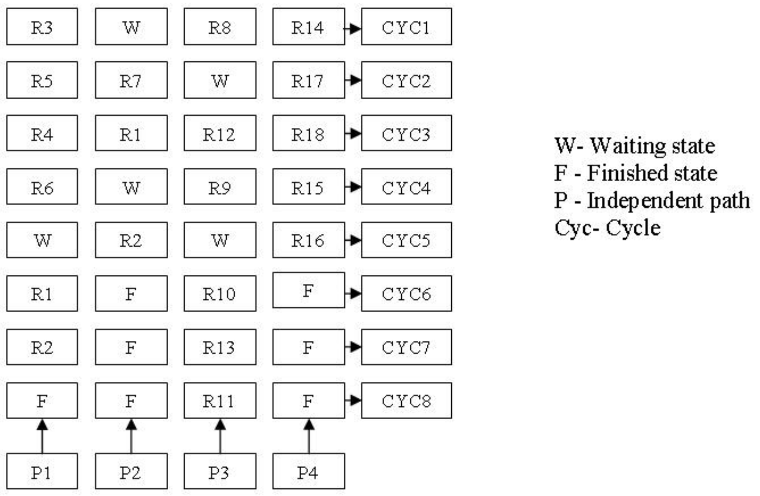


3.7. PTE (PARALLEL TEST CASE EXECUTOR)

RESOURCE POOL:

TEST CASE	RES	TEST CASE	RES
A1-B3-10	R1	A2-B2-20	R10
A1-B4-10	R2	A2-B3-20	R11
A2-B3-10	R3	A3-B2-20	R12
A2-B4-10	R4	A3-B3-20	R13
A3-B3-10	R5	30-B1-10	R14
A3-B4-10	R6	30-B2-10	R15
A1-B2-10	R7	30-B3-10	R16
A1-B2-20	R8	30-B4-10	R17
A1-B3-20	R9		

This contains all combination of Test cases. Each Test case is called Resource. Poll contain following resources. In the parallel executor, each independent path executed by separate processor. Each processor parallely fetches the resource from resource pool if resource is available. If resource is not available at a time of fetching the resources then processor go to waiting state. Once processor completing their task then resource is released and then transfer to resource pool. Now four processor executing following manner.



4. COMPARISON RESULT

1. Overall Test case Calculated by

   (31*51*41)*4

2. Reduced Test case calculated by

   (31*21*1) + (10*41*1) + (31*21*1) + (1*51*1)

3. Assume each test case takes 0.5 seconds.

Overall test case		Reduced test case without parallelism		Reduced test case with parallelism	
No of Test Case	**Execution Time**	**No of Test Case**	**Execution Time**	**No of Test Case**	**Execution Time**
259284	**129642**	**1763**	**881.5**	**1763**	**400**

5. CONCLUSION

The new proposed technique that achieved greater reduction percentage of the test cases and execution time by using parallelism. Furthermore, for compilation, it has been found that the new technique is the least time-consuming among the existing technique. Based on the analysis done, the proposed method can be considered a superior technique from all others available in current literatures. Limitation of the proposed technique is not applicable where there is no common variable in the independent path. The future work on the technique would, therefore, address these problems and find practical measures to overcome them.

6. REFERENCES

[1] B. Beizer. "Software Testing Techniques." Van Nostrand Reinhold, 2nd edition, 1990.

[2] B. Korel, "Automated Software Test Data Generation," Conference on Software Engineering, Vol 10, No. 8, pages 870-879, August 1990.

[3] L. A. Clarke, "A System to Generate Test Data and Symbolically Execute Programs," IEEE Transactions on Software Engineering, Vol. SE-2, No. 3, pages 215-222, September 1976.

[4] L. J. Morell. "A Theory of Error-Based Testing," PhD thesis, University of Maryland, College Park MD, 1984, Technical Report TR-1395

[5] M. J. Gallagher and V. L. Narsimhan, "ADTEST: A Test Data Generation Suite for Ada Software Systems," IEEE Transactions on Software Engineering, Vol. 23, No. 8, pages 473-484, August 1997.

[6] Neelam Gupta, A. P. Mathur and M. L. Soffa,"Automated Test Data Generation using An Iterative Relaxation Method," ACM SIGSOFT Sixth International Symposium on Foundations of Software Engineering (FSE-6),

pages 231-244, Orlando, Florida, November 1998.

[7] Offutt A. Jefferson, J. Pan and J. M. Voas.“Procedures for Reducing the Size of Coverage-based Test.