**Time Aware Test Case Prioritization based on Fault Coverage using Binary Integer Programming**

Sunil
HOD & Assistant Professor, Department of CSE, SSIET, Dinanagar
Dr. Rajesh kumar
Associate professor Department of CSE, Thapar university, Patiala

**ABSTRACT**
Test prioritization schemes typically create a single re-ordering of test cases of the test suite that can be executed after many subsequent changes to the program under test. Test case prioritization techniques reorder the execution of test cases in test suite is an attempt to ensure that defects are revealed earlier in the test execution phase. If testing must be terminated early, a reordered test suite can also be more effective at finding faults than one that was not prioritized. Using the preceding BIP approach, we can select a subset of test cases that can satisfy the time budget and maximize the number of faults covered by the selected test cases. However; there may be some unselected test cases that, if further selected, cannot exceed the time budget. Further selecting such test cases cannot increase the number of faults covered. In this paper discussed about time Aware Test Case Prioritization based on Fault Coverage and the implementation of Fault Coverage using Binary Integer Programming.

**Keywords:** Fault Coverage; Software-Artifact Infrastructure Repository(SIR); Binary integer programming; MATLAB

1. **INTRODUCTION**
Software testing is the key technology for evaluating the fault detecting capability quantitatively and is a vital exercise in quality assurance. Software testing is time consuming, ambiguous, error prone and expensive process. The need of the hour is to provide a cost effective strategy for software testing. Common thread of test cases optimization is formed by test cases classification, minimization, selection and prioritization. Test cases optimization is NP-Complete problem. However, by applying appropriate test case optimization techniques, these efforts and cost can be reduced considerably. Quality of the software testing is directly proportional to the optimization of test cases. Quality of software testing is low due to inadequate strategy for test cases optimization. Binary integer programming for software test cases selection and prioritization with test data adequacy criteria and automation of testing process will help in improving the overall quality of the software. Industry requires cost effective adequate technique for test case selection and prioritization. Fault detection or coverage is prime factor for evaluating adequacy and ranking the test cases. Test cases should prioritize on the basis of fault detection capability. Test cases should be designed, selected and prioritized in such a manner that it will detect the maximum number of faults in software under test within deadline of time and budget. Binary integer programming provides the best solutions to the NP-Complete problems, since test case optimization is NP-Complete problem and can be solved efficiently, accurately using binary integer programming approach. The proposed test case prioritization technique using BIP within a time restricted framework is implemented and evaluated. The technique uses the fault/error coverage and execution time information of the regression test suite as an input. In the proposed algorithm faults coverage acts as a cost of execution. We abbreviate the technique as BIP_TEST.29

The basic block diagram for the Binary Integer Programming for Test Case Selection and Prioritization system (BIP_TEST) is shown in fig 1. The inputs to the system include details of the test suite i.e the test cases along with the faults covered by them and their execution time. These inputs are generally tabulated and are entered by the tester. The produced output has an order of test cases of test suite, finally selected and prioritized.

![Fig: 1.1: Time Aware Test Case Prioritization based on Fault Coverage using BIP](image)

2. **PROBLEM DEFINITION**
Let T be a test suite containing n elements and T= {t1,t2,t3,......tn} be a sequence on T. Let F be a function from test suite elements to some domain on which the relation ≥ imposes a total order, is a test case prioritization of T with respect to F if and only if for all I, 1 ≤ i ≤ n-1 such that F(ti) ≥ F(ti+1), that is F is monotonic over T. ‘T’

A selective regression testing technique chooses a subset of test suite that was used to test the software before modifications were made and then uses this subset to test the modified software. Prioritization is the process of scheduling test cases in a order to meet some performance goal.
We define a test suite \( T \) as a \( n \)-tuple of test cases \( t_i \), where \( i = 1 \) to \( n \) and \( T = \{ t_1, t_2, t_3, \ldots, t_n \} \). The prioritization of \( T \) is denoted as \( \pi \). In the problem formulation, the maximum time within which a prioritize test suite must execute is the maximum capacity of test suite, \( \tau \). Execution time of each test case is its weight and its value is number of faults covered. The output of algorithm is prioritized list that fits the required time limit.

### Illustration with the Example

We have considered that there are six test cases in a regression test suite (denoted as \( T = \{, \} \) ) and there are six errors (denoted as \( F = \{, \} \) ). The test cases and the time required to find out the fault/error are presented in Table 4.1. Let us suppose that the time budget is 19 seconds, we need to find out an ordering (denoted as \( T' \)) of a subset of \( T \) under the condition that the execution time of all selected test cases (sum(time(ti))) is no more than 19 (timemax) seconds. The ordering of test cases can also be effective in errors/faults coverage as early as possible.

### 3. SOLUTION REPRESENTATION

The problem can be represented in the form of an undirected graph \( G (V, E) \) where \( V \) is the set of vertices and \( E \) is the set of edges in the graph. \( \forall \in T \), test cases are represented by vertices in the graph. Each edge in the graph represents the pheromone trail associated with the edge \( e_i E \), which reflects the amount of statement coverage sion the chosen path within time constraint, \( TC. \in \) 3.1 Inputs for Experimental Study

For the experiment, we used printokens2 program, a lexical analyzer, written in c. It is benchmark dataset freely available at website http://www.sir.unl.edu of Software Infrastructure Repository (SIR). Out of the complete test pool, we have randomly chosen a test suite of 500 (five hundred) test cases. Here, we find fault coverage and execution time of each test case. Fault Coveragebility and execution time can be computed using Perl script and time utility of Ubuntu 11.2 and details are as follows:

- **a) Perl Script:** First of all we compile the program printokens2 using gcc gnu compiler. Subsequently, script is run for finding errors/fault coverage for specified test cases. Format of script command is as follows:
  ```
  Source-file-name <Test-case-file-name>Output-file-name
  ```
  Some sample commands of test script for finding the faults coveragebility for test cases () are given below:
  ```
  echo "testing script type: R
  echo "running test 1"
  ../source/print_tokens2.exe < ../inputs/newtst148.tst > ../outputs/t1
  ```

- **b) Time:** time command is used to identify the execution time. Format of this command is as follows:
  ```
  Time [options]command[arguments…]
  ```
  The time command runs the specified program command with the given arguments. When command finishes , time writes a message to standard output giving timing statistics about the program run. These statistics consist of:
  - The elapsed real time between invocation and termination.
  - The user CPU time.
  - The system CPU time.

### 3.2 Program under Test

Printtokens2 version 2 is used for experimentation purpose. It has been taken from Software- Artifact Infrastructure Repository (SIR) and is a lexical analyser. It is a benchmark project and freely available at website www.sir.unl.edu. This program has been implemented in C and consists of 19 procedures, 570 Line of Code (LOC). It has been originally created by Tom Ostrand and colleagues at Siemens Corporate Research and its version is SIRV: 2.0. It consists of 4115 test cases, out of which 500 test cases and 78 faults are used for implementation part.

### 4. IMPLEMENTATION

The algorithm has been named as BIP_TEST and is implemented in MATLAB programming according to the proposed algorithm. Binary Integer Programming tool (BINPROG) is used to select and prioritize the test cases using fault detecting capability with time constraint. We used xlsread() command to read the values of number of faults revealed and execution time of each test cases from Microsoft Excel file named 'fault.xlsx' located in same folder, and store number of faults revealed and execution time of each test cases into one dimensional array \( f \) and \( A \) respectively. Test cases selection and prioritization is done to achieve the maximum fault detection with time constraint of 100 second using BINPROG tool. Under the time constraint environment, program has been compiled and run with the required inputs (number of faults covered and execution time for each test case). Matlab commands for BIP_TEST are given below:

```
'Program for Software test cases selection & prioritization based on faults coverage criteria using Binary Integer Programming,

f=xlsread('fault.xlsx','Sheet1','CA0002:CA501')
faults=f;
faults=faults.*((-1);
A=xlsread('fault.xlsx','Sheet1','CF0002:CF501')
A=A';
b=[100];
X=bintprog(faults,A,b);
```
disp('Test prioritization & Selection are as follows')

5. CONCLUSION
In this paper, have proposed the test case selection and prioritization approach based on BIP to improve fault coverage and reduce the total execution time for regression testing, and obtained the optimal solution in optimum time. Proposed work gave cheering results. BIP is strong and robust as it involves positive feedback and parallel computations and hence, it can lead to better solutions in optimum time. The proposed algorithm is validated by analyzing an industrial project. Result indicates that the proposed technique lead to improved rate of fault detection in comparison to non-prioritized test cases. Multi-objective test case prioritization and selection using Binary Integer programming will be explored in future. We have also plan to further improve our approach using additional coverage rather than resorting to total coverage, when the full coverage is reached, and considering the time and cost required to find the fault when prioritizing the selected test cases. In addition to above, we plan to investigate soft computing techniques for test case selection for time, cost aware prioritization.

6. REFERENCES
[3]. Xiaofang Zhang, Changhai Nie, Baowen Xu and BoQu, “Test Case Prioritization based on Varying Testing Requirement Priorities and Test Case Costs”, Proceedings of Seventh International Conference on Quality Software (QSIC’07), 2007