

Structurally Guided Testing

Harish V. Kantamneni

Sanjay R. Pillai

Yashwant K. Malaiya

Dept of Computer Science, Colorado State University

Email : {harishk, spillai}@microsoft.com, malaiya@cs.colostate.edu

1 Introduction

Black-box testing [2] can be easily automated and involves less processing than white box testing. However it is very hard to achieve high coverage with black-box testing. Hard to test branches have the most impact on the testability of the code. This paper proposes a technique that guides black-box testing so that these *hard to test* branches are covered quickly i.e. with fewer test inputs.

A new dynamic measure termed *potential* of a branch is proposed. During testing, useful structural information is extracted and combined with the changing coverage information to evaluate the current potential of each branch. This measure is used to guide black-box testing techniques so that the new tests generated are more likely to exercise branches which are so far not covered. The results (Table 1) show that this simple guided technique significantly improves coverage, especially for programs with complex structural properties.

2 Background

Voas and Miller [5] defines software testability as “the probability that a piece of software will fail on its next execution during testing (with a particular assumed input distribution) if the software includes a fault”. While testability is a complex issue, it is usually considered as an attribute of a module or a software as a whole. Voas et al [4] give a method to calculate the testability of a module based on the PIE model. The PIE model is a technique that is based on the three part model of software failure. The three necessary and sufficient conditions for a fault to actually cause a failure which is detectable is

Execution : the location where the fault exists or has an impact on must be executed.

Infection : the program data state is changed by the erroneous computation.

Propagation : the erroneous data state is propagated to the program’s output causing an error in the output.

The PIE model uses all three probabilities to calculate the testability of the code. Thus if we can execute locations more often i.e. increase the execution probability, we can improve the testability as calculated by the PIE model.

3 The Guided Approach

Our approach makes it possible to generate tests so as to maximize the branch coverage criterion as quickly as possible. This is primarily a framework for black box testing techniques. The guiding algorithm calculates metrics based on the structure of the branches within the code and uses these metrics to influence the test generation mechanism. Any test generation mechanism could be easily modified to use this metric.

The factors that make a branch hard to cover are (based on random testing of several programs)

1. Nesting factor : Branches that are deeply nested are more difficult to cover. In the potential measure that we have come up with, we quantify this difficulty associated with the nesting factor.
2. Branch Predicates: Branches with certain operators in their predicates are usually harder to cover. We have identified three such operators : *equal to* (=), *not equal to* (!=) and *logical And* (&&).

3.1 The metric : Potential of a branch

It is a dynamic metric that is calculated for each branch after the execution of each test input. Briefly it is the number of branches that are nested within a branch which are yet to be covered. It is a combination of both the nesting level and the number of branches which have not yet been covered. It indicates what the “**potential**” is, i.e. how many new branches is it possible to cover if we can somehow cover the current branch (for which we are calculating this metric). Formally we define it as follows [1]:

Potential of branch i with respect to a test vector is 0 if it has not been covered. If branch i has been covered/entered with respect to the test vector then it's potential is the sum of the number of uncovered branches nested immediately inside this branch i and the potentials of the covered branches nested immediately inside branch i. If all branches within branch i have been covered the potential is 0.

The pseudo-code for calculating the potential of a branch is given below.

```
potential(branch i) # with respect to a vector
begin procedure
  if branch i is covered
    potent = 0
    for x in branches j to k
      #where j to k are branches nested
      #immediately inside branch i
      if (branch x covered by this vector)
        potent = potent + potential(branch x)
      if (branch x has never been covered)
        potent = potent + 1
    return potent
  else
    return 0
end procedure
```

3.2 Description of the algorithm

Initially the test generation technique that has been selected is used without the guiding mechanism, to cover those branches which are easy to cover. After exercising the program with each test vector, the potentials of all the branches in the program under test are calculated and the branch with the highest potential is selected. By focusing the test input generation on this branch, the probability of covering new branches is increased.

The focusing is achieved by input range reduction that localizes the input space about the previous test vector. If there is no change in the potential after several test vectors, then range expansion is carried out so that other promising regions could be identified in

the input space.

4 Results

An experiment was carried out to assess the effectiveness of these techniques. We selected 4 programs [6, 3] of varying sizes from literature or which have standard implementations. They are a mix of program sizes, number of branches, predicate complexity and nesting level. Some of the programs have been slightly modified to allow it to interface properly with our tool. The results are listed in table 1.

Program	No. of Branches	% Coverage Achieved	Test Vectors	
			Random	Guided
triangle	20	100	5527	166
calendar	42	97.62	408	127
roots	41	92.68	3426	1598
max	19	100	8	8

Table 1: Coverage Vs Number of tests needed

The potential metric used in guided testing reduces the impact of the nesting factor on hard to cover branches. A further improvement based on branch predicates [1] is to instrument branches with complex predicates in order to help the guided approach cover these branches faster.

References

- [1] H. V. Kantamneni, S. R. Pillai, and Y. K. Malaiya. Testing software using branch potential. Technical Report 99-102, Computer Science Department, Colorado State University, Fort Collins, CO, 1999.
- [2] Y. K. Malaiya. Antirandom testing: Getting the most out of black-box testing. Technical Report 96-129, Computer Science Department, Colorado State University, Fort Collins, CO, 1996.
- [3] T. R. F. Nonweiler. Roots of low-order polynomial equations. *Communications of the ACM*, 11(4):269, Apr. 1968.
- [4] J. M. Voas. A dynamic testing complexity metric. *Software Quality Journal*, 1(2):101 – 114, June 1992. Chapman and Hall.
- [5] J. M. Voas and K. W. Miller. Software testability: The new verification. *IEEE Software*, May 1995.
- [6] W. E. Wong. *On mutation and dataflow*. PhD thesis, Computer Science Department, Purdue University, 1993.