Sufficient Mutation Operators for Measuring Test Effectiveness

Akbar Siami Namin and James H. Andrews
Department of Computer Science
University of Western Ontario
London, Ontario, Canada
{asiamina, andrews}@csd.uwo.ca

Duncan J. Murdoch
Department of Statistics and Actuarial Sciences
University of Western Ontario
London, Ontario, Canada
murdoch@stats.uwo.ca

30th International Conference on Software Engineering, ICSE 2008, Leipzig, Germany
May 2008
Outline

- Motivation
- A Quick Review of Mutation
- Sufficient Set Problem
- Applicable Statistical Techniques
- Experimental Procedure
- Data Analysis
- Conclusion
- Research Directions
Motivation

- >50% of the development budget is still allocated for testing
- Software systems are becoming more complex

Need More Effective and Less Expensive Testing Techniques

- Measuring the Cost: Easy
  - E.g., # of test cases needed
- Assessing the Effectiveness!!!

How?
Motivation
Assessing the Effectiveness of Testing Techniques

- Empirical Investigation and Experimentation
  - Testing technique A is more effective than testing technique B if A detects more faults than B for a set of subject programs with known faults

Need a set of subject programs accompanied by some faults

Using Mutation to Generate Faulty Versions
**Mutation**

**Terminologies**

- **Mutation** – Modifying the source code slightly
- **Mutation Operator** – Mathematically well-defined functions that transforms an element of the source code to another element in the same class
- **Mutant** – The source code resulting from applying a mutation operator
- **Killed Mutants** – There is at least one test case in the test suite that detects the mutant
  - Running the test case on both the gold and the mutant produces different output
- **Alive Mutants** – No test case in the test suite is able to detect the mutant. But by adding the required test cases it might be detected
- **Equivalent Mutant** – There is absolutely no test case to detect the mutant
Andrews et al. [2005, 2006] showed that:
- Mutants can be a good representation of real faults
- They have high correlations for both random and coverage-based test suites

Mutant ↔ Fault
Mutation
An Expensive Procedure

- > 100 mutation operators
  - E.g., 137 lines of code => 4935 mutants generated by 108 mutation operators
- Makes mutation infeasible

Sufficient Set Problem [Offutt et al., 1996]

“Search for a subset of mutation operators that can be used instead of all mutation operators in measuring the adequacy of test cases”
Sufficient Set Problem

Related Work

- **Mutation Testing**
  - Used to enhance a test suite [Hamlet 1977; De Millo et al. 1978]
  - Mutation Tools
    - Proteum, MuJava

- **Sufficient Operators**
  - Compared 22 operators with 2 operators [Wong 1993]
  - Compared 22 operators with 4 subsets [Offutt et al. 1996]
  - Defined guidelines for selecting [Barbosa et al. 2001]

- **Generating Faulty Versions**
  - Hand-seeding [Frankl and Weiss, 1993]
  - Reintroducing [Frankl and Iakounenko, 1998]
  - Mutation [Daran et al., 1996]
Sufficient Set Problem
Why is Our Work Different?

1. Using larger subject programs with complex data structures
   ▪ 137 – 513 lines of code
2. Sufficient set for a wide range [0, 1] of prediction for mutation score
3. A variable reduction problem
4. Identifying important operators through linear regressions
Sufficient Set Problem
Our Definition

- We refine sufficient set problem as:
  - Finding a subset of mutants and a multiple linear regression model that can predict the behavior of all mutants

The problem is transformed to a statistics problem
For each test suite s, we compute:

1. Mutants Detection Rate (AM) (Generated by all operators):

   \[ AM = \frac{\text{#Killed Non - Equivalent Mutants}}{\text{#All Non - Equivalent Mutants}} \]

2. Mutant Detection Rate (Am_i) (Generated by operator i):

   \[ Am_i = \frac{\text{#Killed Non - Equivalent Mutants of Operator i}}{\text{#All Non - Equivalent Mutants of Operator i}} \]

Sufficient Set Problem
The Scheme
The Problem is to find a linear model:

\[ AM = k + c_1 Am_1 + c_2 Am_2 + ... + c_n Am_n + \varepsilon \]

- The right hand side involves operators with minimum costs
- The prediction should be accurate
Sufficient Set Problem
Cost of Operators

- Sufficient set problem
  - A special case
    - Unequal costs are associated with operators
  - Cost of operator $i$

\[
\text{cost}(i) = \frac{\# \text{Non - Equivalent Mutants Generated by } i}{\# \text{All Non - Equivalent Mutants}}
\]
Possible Statistical Techniques

- What Statistics offers:
  - Variable Reduction Techniques
    - Correlation Analysis
      - Based on the correlation between operators
    - Cluster Analysis
      - Based on the distance of two operators
  - All-Subset Selection and Multiple Linear Models
All Subset Selection

- The Idea of Prediction and Applying Multiple Linear Regression
- Output: A set of good models (active sets: important variables)
- Variants
  - Exhaustive
  - Forward (Backward) Stage-wise
  - Lasso
  - Least Angle Regression
    - Intermediate steps
    - Better performance and answer set
    - Variables with equal costs
Cost-based Least Angle Regression (CbLARS)

- Least Angle Regression (LARS)
  - Variables have equal costs
- Cost-based Least Angle Regression (CbLARS)
  - Developed as a collaboration between two departments (Statistics and Computer Science) at the University of Western Ontario
  - A modification of Original Least Angle Regression
  - Variables are associated with unequal costs
Experimental Procedure

The Artifacts

- **Subject Programs: the Siemens set**
  - Consists of seven programs
    - Net Lines of code: 137 => 513
    - #test cases: 1052 => 5542
- **Mutant Generator: Proteum (Maldonado et al.)**
  - A Mutant generator for C
  - Implements 108 mutation operators
## Experimental Procedure
### A Summary

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Lines of Code</th>
<th>#Mutants</th>
<th>#Selected</th>
<th>#Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcas.c</td>
<td>137</td>
<td>4935</td>
<td>4935</td>
<td>0</td>
</tr>
<tr>
<td>totinfo.c</td>
<td>281</td>
<td>8767</td>
<td>1958</td>
<td>218</td>
</tr>
<tr>
<td>schedule.c</td>
<td>296</td>
<td>4130</td>
<td>1964</td>
<td>204</td>
</tr>
<tr>
<td>schedule2.c</td>
<td>263</td>
<td>6552</td>
<td>1964</td>
<td>467</td>
</tr>
<tr>
<td>printtokens.c</td>
<td>343</td>
<td>11741</td>
<td>1966</td>
<td>415</td>
</tr>
<tr>
<td>printtokens2.c</td>
<td>355</td>
<td>10266</td>
<td>1963</td>
<td>21</td>
</tr>
<tr>
<td>replace.c</td>
<td>513</td>
<td>23847</td>
<td>1969</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>70238</strong></td>
<td><strong>16719</strong></td>
<td></td>
<td><strong>1325</strong></td>
</tr>
</tbody>
</table>

Total #Non-Equivalent Mutants = 15394
Experimental Procedure
Test Suite Generation

Required Coverage Degree = p(req) in \{50, 51, \ldots, 95\}

- \( p(\text{old}) \leftarrow 0 \)
- Randomly select a test case \( t \)
- Add \( t \) to test suite \( s \)
- Compute the new coverage degree \( p(\text{new}) \) of the test suite \( s \)
- Remove \( t \) from \( s \) and \( T \)
- \( P(\text{new}) < P(\text{req}) \)
- \( P(\text{old}) = p(\text{new}) \)
- \( P(\text{new}) > p(\text{old}) \)
- Report \( s \)
Data Analysis
CbLARS – The First Model with

Adjusted $R^2 \geq 0.98$
Data Analysis
CbLARS – Fitted vs. Actual
Data Analysis
CbLARS - CrossValidation
## Data Analysis

### CbLARS – Seven Fold Cross Validation

<table>
<thead>
<tr>
<th>Subjects</th>
<th>MSE</th>
<th>r²</th>
<th>Spearman</th>
<th>Kendall</th>
</tr>
</thead>
<tbody>
<tr>
<td>printtokens.c</td>
<td>0.008</td>
<td>0.930</td>
<td>0.769</td>
<td>0.601</td>
</tr>
<tr>
<td>printtokens2.c</td>
<td>0.003</td>
<td>0.924</td>
<td>0.967</td>
<td>0.849</td>
</tr>
<tr>
<td>replace.c</td>
<td>0.008</td>
<td>0.963</td>
<td>0.973</td>
<td>0.876</td>
</tr>
<tr>
<td>schedule.c</td>
<td>0.003</td>
<td>0.747</td>
<td>0.873</td>
<td>0.725</td>
</tr>
<tr>
<td>schedule2.c</td>
<td>&lt;0.001</td>
<td>0.982</td>
<td>0.911</td>
<td>0.755</td>
</tr>
<tr>
<td>tcas.c</td>
<td>0.022</td>
<td>0.936</td>
<td>0.965</td>
<td>0.866</td>
</tr>
<tr>
<td>totinfo.c</td>
<td>0.015</td>
<td>0.841</td>
<td>0.900</td>
<td>0.765</td>
</tr>
</tbody>
</table>
Data Analysis
CbLARS – Overfitting Check with MSE
Conclusion
Comparison

- CbLARS outperforms others
- The cut-off threshold value for correlation (k = 0.9) might be too high for this problem
- Correlation $\subset$ Cluster
- 22 operators in common

<table>
<thead>
<tr>
<th>Technique</th>
<th># Operators</th>
<th># Mutants</th>
<th>% Mutants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlation</td>
<td>52</td>
<td>6592</td>
<td>42.82%</td>
</tr>
<tr>
<td>Cluster</td>
<td>58</td>
<td>8596</td>
<td>55.83%</td>
</tr>
<tr>
<td>CbLARS</td>
<td>28</td>
<td>1139</td>
<td>7.39%</td>
</tr>
</tbody>
</table>
Conclusion
Interesting Observations

- No Mutation Operators for Mutating Constants Selected!!
  - E.g., Cccr Constant for Constant Replacement
  - Cccr generates many similar mutants
- 50% of negation operators were selected
  - E.g., ArgLogNeg Insert Logical Negation on Argument
Research Directions

- Identifying Sufficient Set for Coverage-based Test Suites
- Sufficient Set for Object-oriented Languages
Thank You