Faster Mutation-based Fault Localization With A Novel Mutation Execution Strategy

Pei Gong, Ruilian Zhao, Zheng Li
Beijing University of Chemical Technology

10th International Workshop on Mutation Analysis (Mutation 2015)
Outline

1. Introduction
2. Faster Mutation-based Fault Localization
3. Empirical Evaluation
4. Conclusion & Future Works
As we know, detecting and finding bugs of software require huge efforts and many researchers paid attentions to fault localization techniques in the past decades.

One of the most popular fault localization techniques is Coverage-based Fault Localization (CBFL), which uses the coverage and test result to estimate the probability that program entities incur error.
Introduction

An Example of CBFL

<table>
<thead>
<tr>
<th>tc₁</th>
<th>tc₂</th>
<th>tc₃</th>
<th>tc₄</th>
<th>tc₅</th>
<th>tc₆</th>
</tr>
</thead>
<tbody>
<tr>
<td>s₁</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>s₂</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>s₃</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>s₄</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>s₅</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>s₆</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>R</td>
<td>P</td>
<td>F</td>
<td>P</td>
<td>P</td>
<td>F</td>
</tr>
</tbody>
</table>

compute the similarity of coverage and test result as *suspiciousness*

suspiciousness formulas

\[
Ochiai = \frac{a_{ef}}{\sqrt{(a_{ef} + a_{nf})(a_{ef} + a_{ep})}}
\]
Mutation Based fault localization (MBFL)

Recently, a Mutation-based Fault Localization is proposed, which combines mutation analysis with fault localization.

Specifically, PUT is executed by a test suite to gather the test cases execution results and coverage of statements.

Then, for each statement covered by failed test cases, a set of mutants are created and rechecked by the previous test suite.
Introduction

Mutation Based fault localization (MBFL)

- The suspiciousness value of each mutant is calculated, and the maximum is set to the corresponding statement as the suspiciousness value of the statement.

- Then, the statements are ranked according to their suspiciousness values from high to low.
## Introduction

### An Example of MBFL

<table>
<thead>
<tr>
<th></th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$s_4$</th>
<th>$s_5$</th>
<th>$s_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$t_2$</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>$t_3$</td>
<td>×</td>
<td>N</td>
<td>K</td>
<td>N</td>
<td>K</td>
<td>✓</td>
</tr>
<tr>
<td>$t_4$</td>
<td>×</td>
<td>K</td>
<td>K</td>
<td>N</td>
<td>K</td>
<td>✓</td>
</tr>
<tr>
<td>$t_5$</td>
<td>✓</td>
<td>N</td>
<td>K</td>
<td>N</td>
<td>K</td>
<td>✓</td>
</tr>
<tr>
<td>$t_6$</td>
<td>✓</td>
<td>K</td>
<td>K</td>
<td>N</td>
<td>K</td>
<td>✓</td>
</tr>
</tbody>
</table>

The suspiciousness of statements 3 is set to maximum suspiciousness of mutants on it. **Suspiciousness**$(s_3)=0.82$

### Suspiciousness Formulas

$$Ochiai = \frac{a_{kf}}{\sqrt{(a_{kf} + a_{nf})(a_{kf} + a_{kp})}}$$

<table>
<thead>
<tr>
<th></th>
<th>suspiciousness</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_1$</td>
<td>0.71</td>
</tr>
<tr>
<td>$m_2$</td>
<td>0.50</td>
</tr>
<tr>
<td>$m_3$</td>
<td>0.82</td>
</tr>
<tr>
<td>$m_4$</td>
<td>0.50</td>
</tr>
<tr>
<td>$m_5$</td>
<td>0.82</td>
</tr>
<tr>
<td>$m_6$</td>
<td>0.71</td>
</tr>
<tr>
<td>$m_7$</td>
<td>0.33</td>
</tr>
</tbody>
</table>

**Suspicousness** of mutants and faulty program as **suspiciousness** compute the similarity of mutants and faulty program as **suspiciousness**
1 Introduction

An Example of CBFL

<table>
<thead>
<tr>
<th></th>
<th>$tc_1$</th>
<th>$tc_2$</th>
<th>$tc_3$</th>
<th>$tc_4$</th>
<th>$tc_5$</th>
<th>$tc_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>$s_2$</td>
<td>✔</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>✔</td>
</tr>
<tr>
<td>$s_3$</td>
<td>×</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>×</td>
</tr>
<tr>
<td>$s_4$</td>
<td>×</td>
<td>×</td>
<td>✔</td>
<td>×</td>
<td>✔</td>
<td>×</td>
</tr>
<tr>
<td>$s_5$</td>
<td>×</td>
<td>✔</td>
<td>×</td>
<td>✔</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>$s_6$</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

compute the similarity of coverage and test result as *suspiciousness*

$s_1$ | 0.58 | 3  
$s_2$ | 0.00 | 6  
$s_3$ | 0.71 | 1  
$s_4$ | 0.50 | 5  
$s_5$ | 0.50 | 5  
$s_6$ | 0.58 | 3  

R P F P P F P  

$s$  

$suspiciousness$  

$rank$  

$suspiciousness$ formulas

\[
Ochiai = \frac{a_{ef}}{\sqrt{(a_{ef} + a_{nf})(a_{ef} + a_{ep})}}
\]
Introduction

- It has been shown that the MBFL is more precise than CBFL.

- However the mutation analysis also brings huge execution cost, since MBFL need to run every test case on each mutant.
This paper focuses on reducing the cost of MBFL by dynamically prioritizing the mutants and test cases that can contribute higher suspiciousness value.

A Dynamic Mutation Execution Strategy (DMES) is proposed, which contains execution optimizations on both mutants and test cases.

As fewer mutants and test cases are executed with DMES, the whole process will become faster and the cost will be decreased.
Faster MBFL

Motivation

- MBFL only pays attention to the mutants with maximum suspiciousness, so the mutants with low suspiciousness are not necessary to be executed.

- If the executions on mutants with low suspiciousness could be reduced, the effectiveness of MBFL can be improved.

Dynamic Mutation Execution Strategy (DMES) contains execution optimizations on both mutants and test cases.
The object of MEO strategy is to skip the execution of mutants with lower of suspiciousness value than the current maximum.

Since the suspiciousness value cannot be obtained without execution of the mutant, the key issue is to estimate the upper boundary of the suspiciousness value for a mutant by running only few test cases.
In general, failed test cases are only small portion of test suite, they usually have a larger impact to fault localization than passed test cases.

So we run $T_f$ on all mutants first, and use their results to compute the upper bound of mutant’s suspiciousness value.
Mutation Execution Optimization strategy (MEO)

- Using Ochiai formula as an example

\[
Ochiasi = \frac{a_{kf}}{\sqrt{(a_{kf} + a_{nf})(a_{kf} + a_{kp})}}
\]

- After running failed test cases, the value of \(a_{kf}\) and \(a_{nf}\) are known.

- If \(a_{kp} = 0\), the suspiciousness value is the biggest.
The upper boundary can be calculated by setting $a_{kp}$ to 0, meaning $a_{np}$ to the number of passed test cases $|T_p|$

So, upper boundary of mutant $m$’s suspiciousness can be computed as:

$$Sus(m) = \frac{a_{kf}}{\sqrt{(a_{kf} + a_{nf}) \times a_{kf}}}$$
The upper boundary can be calculated by setting $a_{kp}$ to 0 and $a_{np}$ to the number of passed test cases $|T_p|$

So, upper boundary of mutant $m$’s suspiciousness can be computed as:

$$\overline{Sus(m)} = \frac{a_{kf}}{\sqrt{(a_{kf} + a_{nf}) \times a_{kf}}}$$

If $\overline{Sus(m)}$ is lower than the current maximal suspiciousness (denoted as $cur_{max}$), $m$ can be skipped in the following execution (passed test cases execution).

Furthermore, if mutants are executed in the order of $\overline{Sus(m)}$ decreasing and $\overline{Sus(m)}$ is lower than $cur_{max}$, all the following mutants can be skipped.
Faster MBFL

Mutation Execution Optimization strategy (MEO)

Framework of MEO

1. Input a statement $s$ in $Cov(T_j)$
2. Mutate $s$ to generate a set of mutants $M(s)$
3. Run $T_j$ on all mutants in $M(s)$
4. Calculate upper bound of suspiciousness value $Sus(m)$ for all mutants in $M(s)$
5. Prioritize mutants according to $Sus(m)$
6. Run $T_j$ on the mutant $m$ sequentially, calculate $Sus(m)$ and update $cur_{max}$
7. Termination condition
   - NO
   - YES
   - Output the suspiciousness value of $s Sus(s)$
Faster MBFL

**Mutation Execution Optimization strategy (MEO)**

Framework of MEO

For a statement $s$ covered by failed test cases, generate mutants for $s$, that is the set of $M(s)$. 

1. Input a statement $s$ in $Cov(T_f)$
2. Mutate $s$ to generate a set of mutants $M(s)$
3. Run $T_f$ on all mutants in $M(s)$
4. Calculate upper bound of suspiciousness value $Sus(m)$ for all mutants in $M(s)$
5. Prioritize mutants according to $Sus(m)$
6. Run $T_f$ on the mutant $m$ sequentially, calculate $Sus(m)$ and update $cur_{max}$
7. Termination condition
   - NO
   - YES: Output the suspiciousness value of $Sus(s)$
Faster MBFL

Mutation Execution Optimization strategy (MEO)

Framework of MEO

- Input a statement $s$ in $Cov(T_f)$
- Mutate $s$ to generate a set of mutants $M(s)$
- Run $T_f$ on all mutants in $M(s)$
- Calculate upper bound of suspiciousness value $Sus(m)$ for all mutants in $M(s)$
- Prioritize mutants according to $Sus(m)$
- Run $T_f$ on the mutant $m$ sequentially, calculate $Sus(m)$ and update $cur_{max}$
- Termination condition
  - YES
  - Output the suspiciousness value of $s Sus(s)$
  - NO
- Failed test cases are executed on all mutants for calculating their upper bound of suspiciousness.
Faster MBFL

Mutation Execution Optimization strategy (MEO)

Framework of MEO

Input a statement $s$ in $Cov(T_i)$

Mutate $s$ to generate a set of mutants $M(s)$

Run $T_i$ on all mutants in $M(s)$

Calculate upper bound of suspiciousness value $Sus(m)$ for all mutants in $M(s)$

Prioritize mutants according to $Sus(m)$

Run $T_i$ on the mutant $m$ sequentially, calculate $Sus(m)$ and update $cur_{max}$

Termination condition

YES

Output the suspiciousness value of $s$ $Sus(s)$

NO

Prioritize mutants based on their $Sus(m)$.

Run passed test cases on mutant $m$ sequentially, calculate its suspiciousness value $Sus(m)$ and update $cur_{max}$.

Execute mutant one by one until upper bound of a mutant lower than $cur_{max}$, then terminate following executions and return $cur_{max}$ as $Sus(s)$. 
However, even \( \overline{Sus(m)} \geq cur_{max} \), \( Sus(m) \) may still be less than \( cur_{max} \).

For such mutants, if the relation of \( Sus(m) \leq cur_{max} \) can be determined without running all test cases, the computational cost of \( Sus(m) \) can be reduced.

Test Cases Execution Optimization strategy is proposed, which focuses on reducing the execution of unnecessary test cases.
Test cases Execution Optimization strategy (TEO)

- How to identify $Sus(m) \leq cur_{max}$ by running as few passed test cases as possible?

- Firstly, the *suspiciousness* of $m$ has a negative correlation with $a_{kp}$. (Mutant killed by more passed test cases, the *suspiciousness* is lower.)

- Using Ochiai formula as an example again

\[
\text{suspiciousness formulas} \\
Ochiai = \frac{a_{kf}}{\sqrt{(a_{kf} + a_{nf})(a_{kf} + a_{kp})}}
\]

The larger is $a_{kp}$, the *suspiciousness* is smaller.
So there is a \textit{threshold} for \( m \) to make \( \text{Sus}(m) \leq \text{cur}_{\text{max}} \). Once \( a_{kp} \) exceeds \textit{threshold}, the executions of the rest test cases on \( m \) can be cancelled.

\textit{Threshold} could be calculated as:

\[
\text{threshold} = \begin{cases} 
|T_p|, & \text{if } \text{cur}_{\text{max}} = 0; \\
\frac{a_{kf}^2}{\text{cur}_{\text{max}}^2 \cdot |T_f|} - akf, & \text{if } \text{cur}_{\text{max}} \neq 0.
\end{cases}
\]
Once the current number of passed test cases killing $m$, $cur_{kp}$, exceeds threshold, the rest passed test cases execution on $m$ could be cancelled.

If $cur_{kp}$ increases more quickly, the execution could be terminated earlier. So, passed test cases that could kill $m$ should be executed on $m$ earlier.

In general, if a test case is capable of killing a mutant, it possibly kills other mutants located on the same statement.

So the passed test cases could be executed according to the number of mutants that they already killed, namely history. A passed test case that has killed more mutants should be run earlier.
Test cases Execution Optimization strategy (TEO)

1. For a mutant $m$ in the order, calculate threshold and prioritize the execution sequence of test cases in $T_p$ by history.
2. Run the test case $t$ sequentially on $m$, count $cur_p$ and update history.
3. Are all test Cases in $T_p$ are executed on $m$?
   - NO: Go back to step 2.
   - YES: Proceed to the next step.
4. $cur_p \geq$ threshold?
   - NO: Go back to step 2.
   - YES: Proceed to the next step.
5. Calculate $Sus(m)$ and update $cur_{max}$. 


For a mutant $m$ in the order decided by MEO, the *threshold* is calculated, and passed test cases are prioritized based on the *history* information.

Then passed test cases are executed on $m$ one by one, with counting $cur_{kp}$ and updating *history*. 
If we can confirm $cur_{kp} \geq threshold$, the rest passed test case executions should be cancelled.

If all passed test cases are executed on $m$, we can calculate suspiciousness of $m$ and update $cur_{max}$ as MEO do.
The DMES is combined with the above two optimizations:

- In statement level, using MEO to prioritize the mutant executions of every statement, and skip the execution of mutants with lower suspiciousness than current maximum.

- In mutant level, using TEO to prioritize the test case executions on mutant, and cancel the executions of rest test cases while the mutant is identified to be without maximum suspiciousness.
Empirical Evaluation

Research Questions

- **RQ1**: Whether Faster-MBFL has precision loss with comparing with MBFL and is more accurate than Coverage-based approach?

- **RQ2**: How does the mutation execution cost of Faster-MBFL with comparing to MBFL?

- **RQ3**: How much additional run time is required by using the presented strategies?
127 faulty versions of six programs are used as programs under test.
The number in the parentheses is the actual used fault versions.
suspiciousness Formulas

<table>
<thead>
<tr>
<th>Name</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarantula</td>
<td>[ \frac{a_k f}{a_k f + a_n f + a_{kp} + a_{np}} ]</td>
</tr>
<tr>
<td>Ochiai</td>
<td>[ \frac{a_k f}{\sqrt{(a_k f + a_n f)(a_k f + a_{kp})}} ]</td>
</tr>
<tr>
<td>Op2</td>
<td>[ a_{kp} - \frac{a_{np}}{a_{kp} + a_{np} + 1} ]</td>
</tr>
</tbody>
</table>

CBFL, MBFL and Faster-MBFL

<table>
<thead>
<tr>
<th>Short Name</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CB_TA</td>
<td>Using Tarantula formula</td>
</tr>
<tr>
<td>CB_OC</td>
<td>Using Ochiai formula</td>
</tr>
<tr>
<td>CB_OP</td>
<td>Using Op2 formula</td>
</tr>
<tr>
<td>MB_TA</td>
<td>Using Tarantula formula</td>
</tr>
<tr>
<td>MB_OC</td>
<td>Using Ochiai formula</td>
</tr>
<tr>
<td>MB_OP</td>
<td>Using Op2 formula</td>
</tr>
<tr>
<td>MEO_TA</td>
<td>Using MEO and Tarantula formula</td>
</tr>
<tr>
<td>MEO_OC</td>
<td>Using MEO and Ochiai formula</td>
</tr>
<tr>
<td>MEO_OP</td>
<td>Using MEO and Op2 formula</td>
</tr>
<tr>
<td>DMES_TA</td>
<td>Using MEO, TEO and Tarantula formula</td>
</tr>
<tr>
<td>DMES_OC</td>
<td>Using MEO, TEO and Ochiai formula</td>
</tr>
<tr>
<td>DMES_OP</td>
<td>Using MEO, TEO and Op2 formula</td>
</tr>
</tbody>
</table>
RQ1: Whether Faster-MBFL, has precision loss with comparing with MBFL and is more accurate than Coverage-based approach?

The *Score* metric is used to evaluate the fault localization ability.

\[
Score = \frac{rank}{\text{NumberOfExecutedStatement}} \times 100\%
\]

*rank* denotes the number of statements that need to be inspected before finding all faulty statements.

*Score* metric can measure the human effort while using the FL techniques, so the lower *Score* means more precision in fault localization.
## Empirical Evaluation

### Result and Analysis for RQ1

**TABLE IV. LOCALIZATION EFFECTIVENESS COMPARISON OF MBFL AND FASTER-MBFL**

<table>
<thead>
<tr>
<th>Score (≤)</th>
<th>MB_TA</th>
<th>MEO_TA</th>
<th>MB_OC</th>
<th>MEO_OC</th>
<th>MB_OP</th>
<th>MEO_OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1%</td>
<td>0.244</td>
<td>0.244</td>
<td>0.378</td>
<td>0.378</td>
<td>0.315</td>
<td>0.315</td>
</tr>
<tr>
<td>5%</td>
<td>0.646</td>
<td>0.646</td>
<td>0.772</td>
<td>0.772</td>
<td>0.661</td>
<td>0.661</td>
</tr>
<tr>
<td>10%</td>
<td>0.795</td>
<td>0.795</td>
<td>0.858</td>
<td>0.858</td>
<td>0.740</td>
<td>0.740</td>
</tr>
<tr>
<td>15%</td>
<td>0.866</td>
<td>0.866</td>
<td>0.890</td>
<td>0.890</td>
<td>0.772</td>
<td>0.772</td>
</tr>
<tr>
<td>20%</td>
<td><strong>0.913</strong></td>
<td><strong>0.913</strong></td>
<td><strong>0.898</strong></td>
<td><strong>0.898</strong></td>
<td>0.835</td>
<td>0.835</td>
</tr>
<tr>
<td>30%</td>
<td>0.961</td>
<td>0.961</td>
<td>0.976</td>
<td>0.976</td>
<td>0.906</td>
<td>0.906</td>
</tr>
<tr>
<td>40%</td>
<td>0.976</td>
<td>0.976</td>
<td>0.976</td>
<td>0.976</td>
<td>0.937</td>
<td>0.937</td>
</tr>
<tr>
<td>50%</td>
<td>0.984</td>
<td>0.984</td>
<td>0.984</td>
<td>0.984</td>
<td>0.953</td>
<td>0.953</td>
</tr>
<tr>
<td>60%</td>
<td>0.992</td>
<td>0.992</td>
<td><strong>0.992</strong></td>
<td><strong>0.992</strong></td>
<td><strong>1.000</strong></td>
<td><strong>1.000</strong></td>
</tr>
<tr>
<td>70%</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>80%</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>90%</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>100%</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Faster-MBFL has the same precision as MBFL with same suspicious formula.

Ochiai formula has best precision in both MBFL and Faster-MBFL except at 20% and 60%.
The fault localization ability of Faster-MBFL is compared with CBFL.

Each Faster-MBFL is better than every CBFL on fault localization precision.
RQ2: How does the mutation execution cost of Faster-MBFL compare to MBFL?

The number of Mutant-Test Pair (MTP) executed is used to measure the mutation execution cost.
Empirical Evaluation

Result and Analysis for RQ2

For limited space, here is the results of MEO, DMEO, and MBFL using Ochiai formula (MEO_OC, OMEO_OC and DMES_OC compared to MB_OC)

is the average MTP executions reduced by using MEO strategy in MBFL.

is mapped to the average MTP executions further reduced by using TEO on MEO.
Empirical Evaluation

Result and Analysis for RQ2

For limited space, here is the results of MEO_OC and DMES_OC compared to MB_OC.

From Figure 4, we can find that, in most cases, the MEO and TEO can significantly reduce the MTP execution of MBFL.
But there are still some special cases.

There are two versions, which MTPs are much lower than others. The reason is that, these two versions only have 3 and 2 failed test cases respectively and few statements are covered by them.
Average reduction rates of MEO and DMES for all faulty programs are about 42.6% and 65% respectively.
Randomly initial execution sequences of mutants and test cases may impact the effectiveness of MEO and DMES, the experiments are repeated 10 times. The biggest difference between Max and Min does not larger than 20% of average. So we believe that randomly initial execution sequences have little influence to the effectiveness of our strategies.
RQ3: How much additional run time is required by using the presented strategies?

The actual run time of using DMES is used by excluding the mutation execution time.
The additional run time is recorded to evaluate the overhead of MEO and DMES.

Even the biggest additional run time does not exceed 11 second.

So, with reducing much mutation execution cost, the additional cost of using MEO and DMES is acceptable.
This paper presents a DMES, which includes two optimizations respectively for executing mutants and test cases.

By dynamic prioritizing the execution sequences, DMES can avoid the executions that test cases on mutants with low suspiciousness.

The empirical studies suggest that both two optimizations can effectively reduce the mutation execution cost of MBFL without fault localization precision loss. In addition, the additional run time of using DMES also can be ignored.
Q&A

Thank you for your attention!