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

- 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

suspiciousness

An Example of CBFL

	tc_1	tc_2	tc_3	tc_4	tc_5	tc_6
s_1	✓	✓	✓	✓	✓	✓
s_2	✓	×	×	×	×	✓
s_3	×	✓	✓	✓	✓	×
s_4	×	×	✓	×	✓	×
s_5	×	✓	×	✓	×	×
s_6	✓	✓	✓	✓	✓	✓
R	P	F	P	P	F	P

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

→

suspiciousness formulas

	suspiciousness	*rank*
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

$$Ochiai = \frac{a_{ef}}{\sqrt{(a_{ef} + a_{nf})(a_{ef} + a_{ep})}}$$

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.

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.

1 Introduction

An Example of MBFL

	tc_1	tc_2	tc_3	tc_4	tc_5	tc_6
s_3	✗	✓	✓	✓	✓	✗
R	P	×	P	P	×	P
m_1	-	K	N	N	N	-
m_2	-	N	K	N	K	-
m_3	-	K	K	N	K	-
m_4	-	N	K	N	K	-
m_5	-	K	K	N	K	-
m_6	-	K	N	N	N	-
m_7	-	K	K	K	N	-

compute the similarity of mutants and faulty program as *suspiciousness*

suspiciousness formulas

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

	suspiciousness
m_1	0.71
m_2	0.50
m_3	**0.82**
m_4	0.50
m_5	**0.82**
m_6	0.71
m_7	0.33

***Suspiciousness* of statements 3 is set to maximum *suspiciousness* of mutants on it.**

$Suspiciousness(s_3)=0.82$

1 Introduction

An Example of CBFL

	tc_1	tc_2	tc_3	tc_4	tc_5	tc_6
s_1	✓	✓	✓	✓	✓	✓
s_2	✓	×	×	×	×	✓
s_3	×	✓	✓	✓	✓	×
s_4	×	×	✓	×	✓	×
s_5	×	✓	×	✓	×	×
s_6	✓	✓	✓	✓	✓	✓
R	P	F	P	P	F	P

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

suspiciousness formulas

	suspiciousness	*rank*
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

$$Ochiai = \frac{a_{ef}}{\sqrt{(a_{ef} + a_{nf})(a_{ef} + a_{ep})}}$$

1 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.

1 Introduction

- 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.

2 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.

2 Faster MBFL

Mutation Execution Optimization strategy (MEO)

- 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.

2 Faster MBFL

Mutation Execution Optimization strategy (MEO)

- 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.

2 Faster MBFL

Mutation Execution Optimization strategy (MEO)

- Using Ochiai formula as a example

suspiciousness formulas

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

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

- If a_{kp}=0, the suspiciousness value is the bigest.

2 Faster MBFL

Mutation Execution Optimization strategy MEO)

- 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:

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

2 Faster MBFL

Mutation Execution Optimization strategy MEO)

- The upper boundary can be calculated by setting a_{kp} to 0 and a_{np} to the number of passed test cases $|Tp|$

- So, upper boundary of mutant m's suspiciousness can be computed as:

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

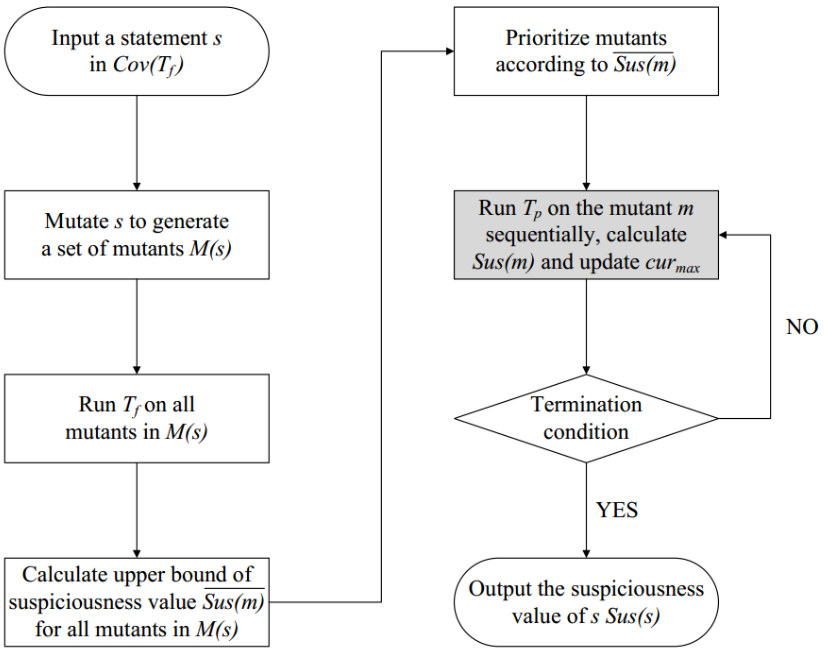
- If $\overline{Sus(m)}$ is lower than the current maximal *suspiciousness* (denoted as cur_{max}), m can be skipped in the following execution (passed test cases execution).

- 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.

2 Faster MBFL

Mutation Execution Optimization strategy (MEO)

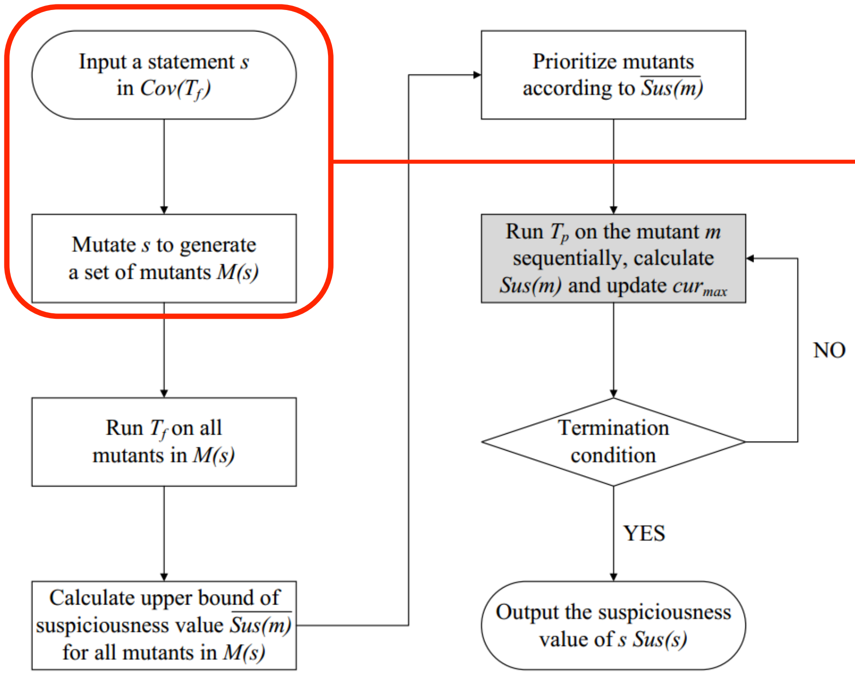
Framework of MEO



2 Faster MBFL

Mutation Execution Optimization strategy (MEO)

Framework of MEO

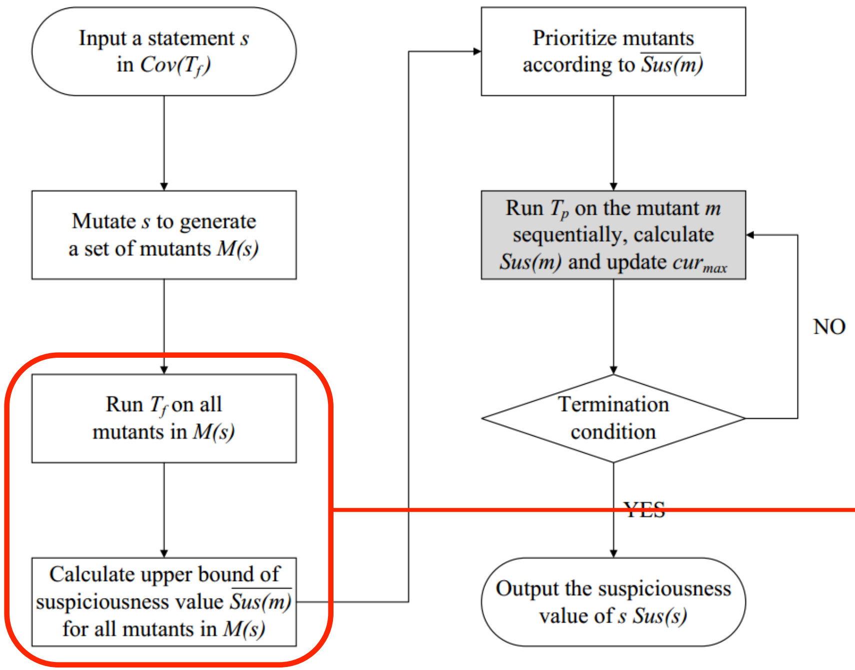


For a statement *s* covered by failed test cases, generate mutants for *s*, that is the set of *M(s)*.

2 Faster MBFL

Mutation Execution Optimization strategy (MEO)

Framework of MEO

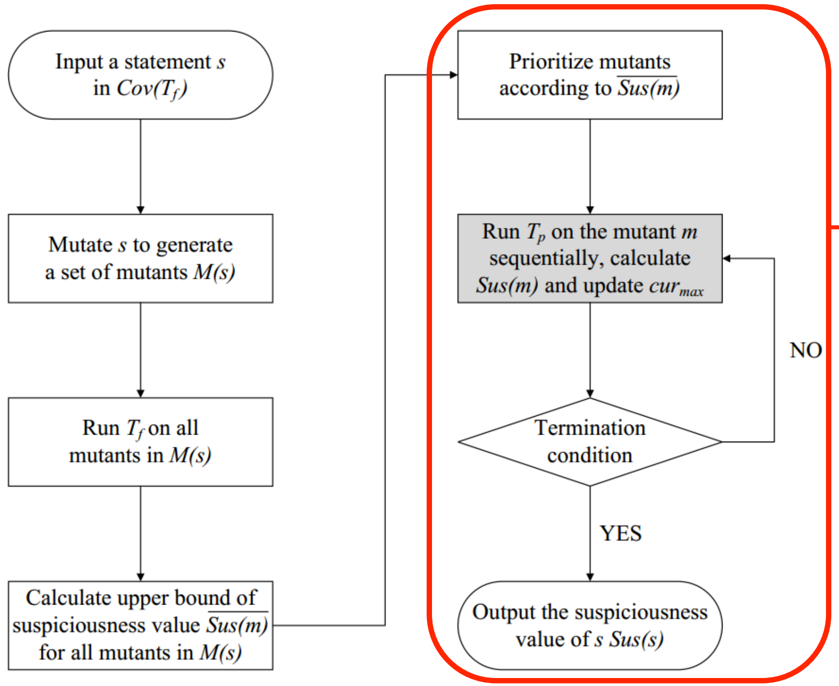


Failed test cases are executed on all mutants for calculating their upper bound of *suspiciousness*.

2 Faster MBFL

Mutation Execution Optimization strategy (MEO)

Framework of MEO



Prioritize mutants based on their $\overline{Sus(m)}$.

Run passed test cases on mutant *m* sequential, 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)*.

2 Faster MBFL

- 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.

2 Faster MBFL

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 a example again

suspiciousness formulas

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

The larger is a_{kp}, the *suspiciousness* is smaller.

2 Faster MBFL

Test cases Execution Optimization strategy (TEO)

- So there is a *threshold* for m to make $Sus(m) \leq cur_{max}$. Once a_{kp} exceeds *threshold*, the executions of the rest test cases on m can be cancelled.

- *Threshold* could be calculated as:

$$threshold = \begin{cases} |T_p|, & \textbf{if } cur_{max} = 0; \\ \left\lceil \dfrac{{a_{kf}}^2}{{cur_{max}}^2 * |Tf|} \right\rceil - akf, & \textbf{if } curmax \neq 0. \end{cases}$$

2 Faster MBFL

Test cases Execution Optimization strategy (TEO)

- 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.

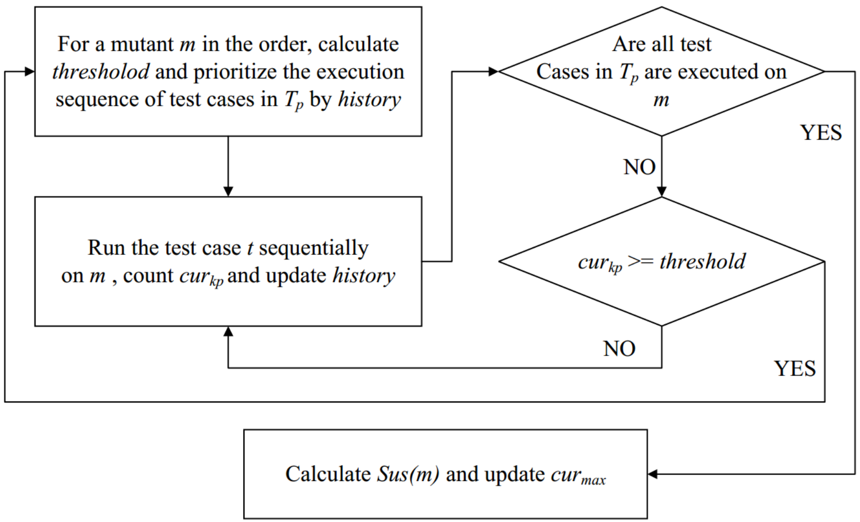
- If cur_{kp} increases more quickly, the execution could be terminated earlier. So, passed test cases that could kill *m* should be executed on *m* earlier

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

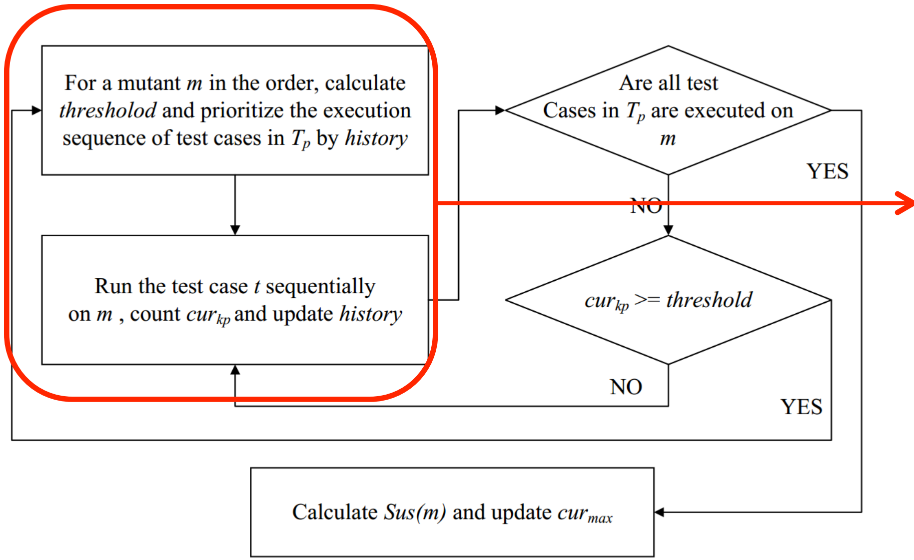
- So the passed test cases could be executed according to the number of mutants that they already killed, namely *history*. A passed test case that has killed more mutants should be run earlier.

2 Faster MBFL

Test cases Execution Optimization strategy (TEO)



Test cases Execution Optimization strategy (TEO)

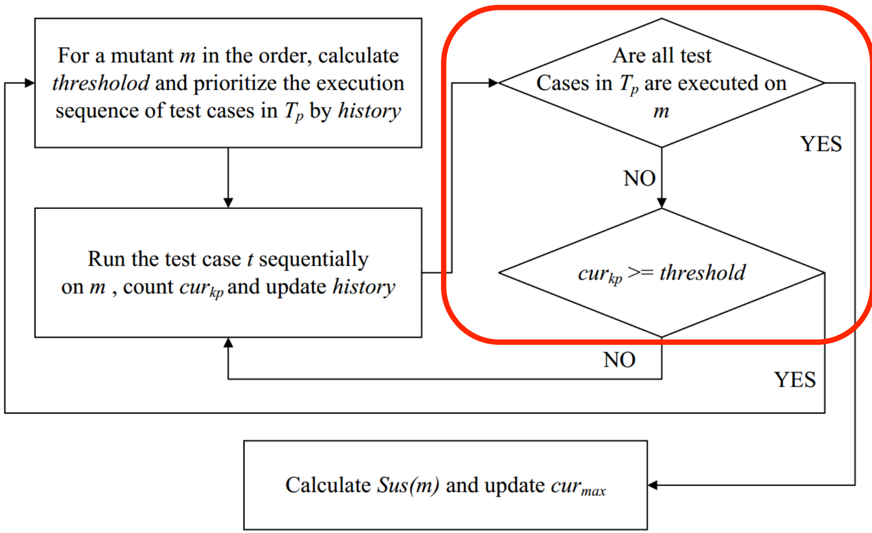


For a mutant *m* in the order decided by MEO, the *threshold* is calculated, and passed test cases are prioritized based on the *history* information.

Then passed test cases are executed on *m* one by one, with counting cur_{kp} and updating *history*.

2 Faster MBFL

Test cases Execution Optimization strategy (TEO)



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.

2 Faster MBFL

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*.

3 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?

Experiment Regimes

Subject Programs

Subject Program	Number of Faulty Versions	Lines of Code	Size of Test Suite	Average Number of Mutants
printtokens	7 (3)	341-343	4130	4249
schedule	9 (5)	290-294	2650	2225
tcas	41 (40)	133-137	1608	5031
totinfo	23 (23)	272-273	1052	6376
replace	32 (28)	508-515	5542	10825
space	38 (28)	5882-5904	13585	38830

150(127)

127 faulty versions of six programs are used as programs under test.

The number in the parentheses is the actual used fault versions

Experiment Regimes

suspiciousness Formulas

Name	Formula
Tarantula	$\frac{\frac{a_{kf}}{a_{kf}+a_{nf}}}{\frac{a_{kf}}{a_{kf}+a_{nf}}+\frac{a_{kp}}{a_{kp}+a_{np}}}$
Ochiai	$\frac{a_{kf}}{\sqrt{(a_{kf}+a_{nf})(a_{kf}+a_{kp})}}$
Op2	$a_{kf}-\frac{a_{kp}}{a_{kp}+a_{np}+1}$

CBFL, MBFL and Faster-MBFL

	Short Name	Descriptions
CBFL	CB_TA	Using Tarantula formula
	CB_OC	Using Ochiai formula
	CB_OP	Using Op2 formula
MBFL	MB_TA	Using Tarantula formula
	MB_OC	Using Ochiai formula
	MB_OP	Using Op2 formula
Faster-MBFL	MEO_TA	Using MEO and Tarantula formula
	MEO_OC	Using MEO and Ochiai formula
	MEO_OP	Using MEO and Op2 formula
	DMES_TA	Using MEO,TEO and Tarantula formula
	DMES_OC	Using MEO,TEO and Ochiai formula
	DMES_OP	Using MEO,TEO and Op2 formula

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.

$$Score = \frac{rank}{NumberOfExecutedStatement} * 100\%$$

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.

3 Empirical Evaluation

Result and Analysis for RQ1

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

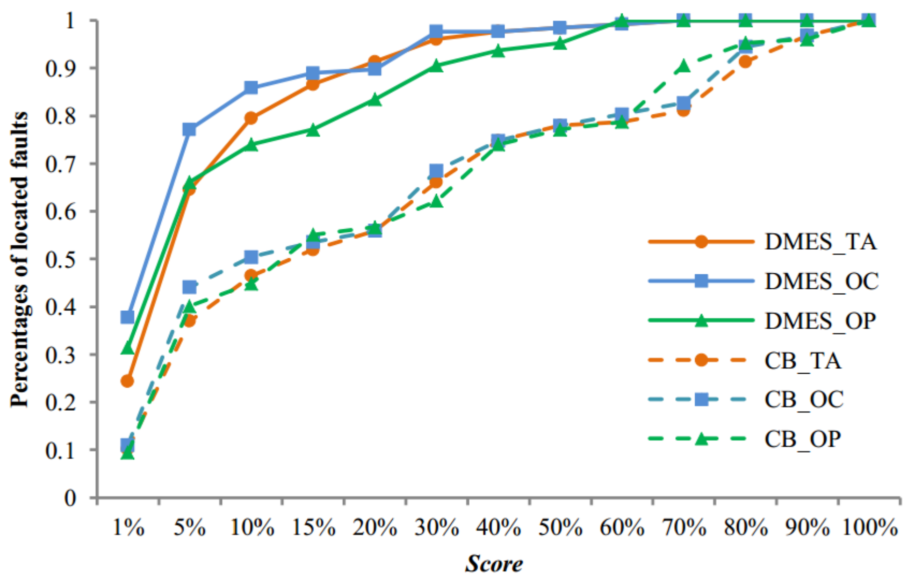
Score (≤)	MB_TA	MEO_TA DMES_TA	MB_OC	MEO_OC DMES_OC	MB_OP	MEO_OP DMES_OP
1%	0.244	0.244	0.378	0.378	0.315	0.315
5%	0.646	0.646	0.772	0.772	0.661	0.661
10%	0.795	0.795	0.858	0.858	0.740	0.740
15%	0.866	0.866	0.890	0.890	0.772	0.772
20%	**0.913**	**0.913**	**0.898**	**0.898**	0.835	0.835
30%	0.961	0.961	0.976	0.976	0.906	0.906
40%	0.976	0.976	0.976	0.976	0.937	0.937
50%	0.984	0.984	0.984	0.984	0.953	0.953
60%	0.992	0.992	**0.992**	**0.992**	**1.000**	**1.000**
70%	1.000	1.000	1.000	1.000	1.000	1.000
80%	1.000	1.000	1.000	1.000	1.000	1.000
90%	1.000	1.000	1.000	1.000	1.000	1.000
100%	1.000	1.000	1.000	1.000	1.000	1.000

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%.

Result and Analysis for RQ1

The fault localization ability of Faster-MBFL is compared with CBFL.



Localization Effectiveness Comparison of SBFL and Faster-MBFL

Each Faster-MBFL is better than every CBFL on fault localization precision.

3 Empirical Evaluation

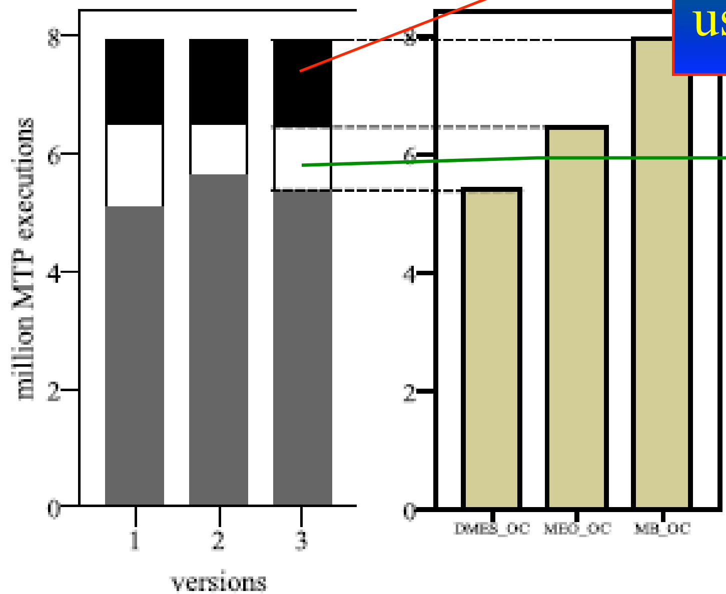
RQ2：How does the mutation execution cost of Faster-MBFL with comparing to MBFL?

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

3 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)



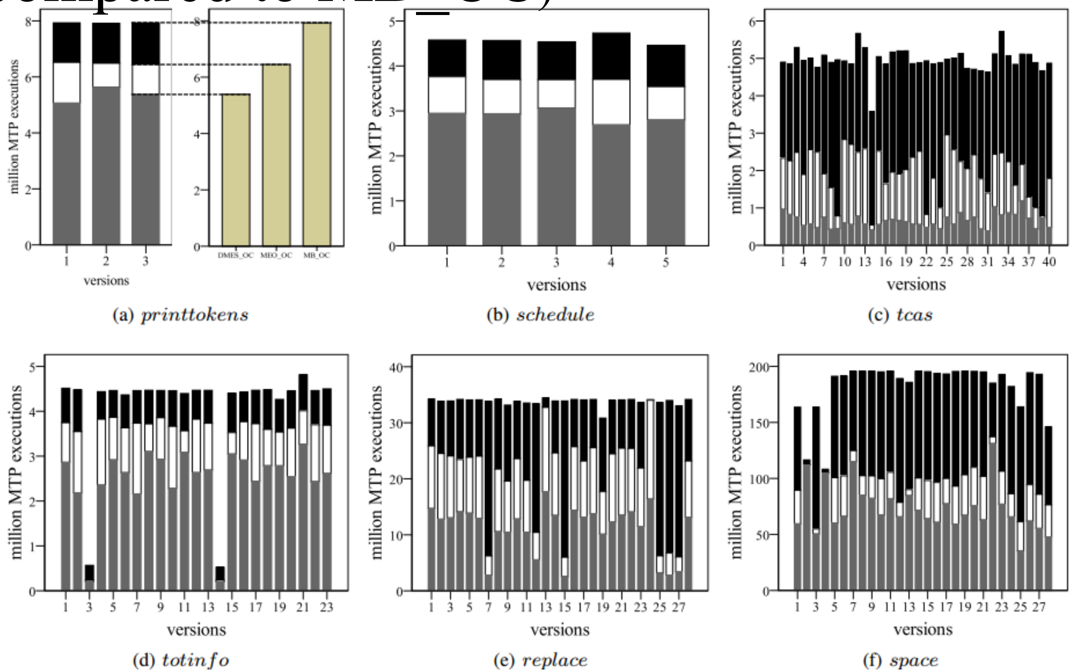
is the average MTP executions reduced by using MEO strategy in MBFL.

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

3 Empirical Evaluation

Result and Analysis for RQ2

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

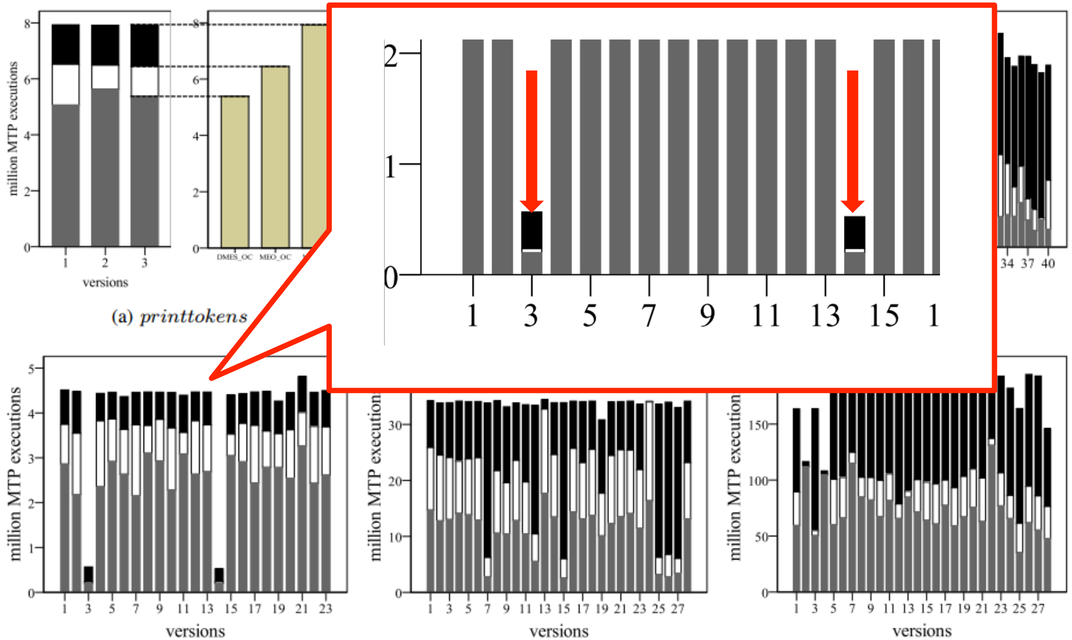


From Figure 4, we can find that, in most cases, the MEO and TEO can significantly reduce the MTP execution of MBFL.

3 Empirical Evaluation

Result and Analysis for RQ2

But there are still some special cases.



(a) *printtokens*

(d) *totinfo*

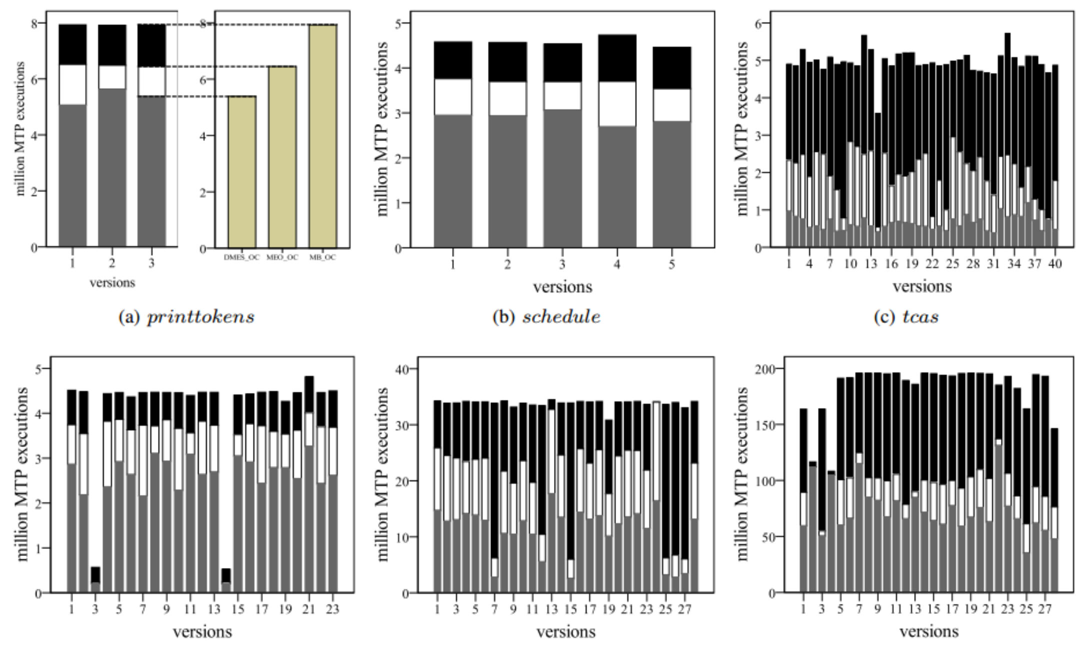
(e) *replace*

(f) *space*

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.

3 Empirical Evaluation

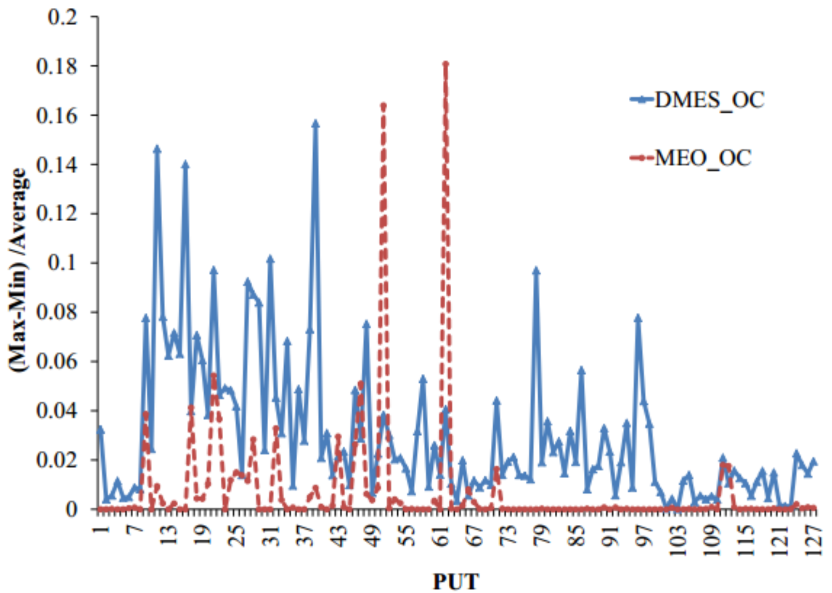
Result and Analysis for RQ2



Average reduction rates of MEO and DMES for all faulty programs are about 42.6% and 65% respectively.

Result and Analysis for RQ2

Randomly initial execution sequences of mutants and test cases may impact the effectiveness of MEO and DMES, the experiments are repeated 10 times.



The Ratio of MTP Execution ranges by MEO OC and DMES OC

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.

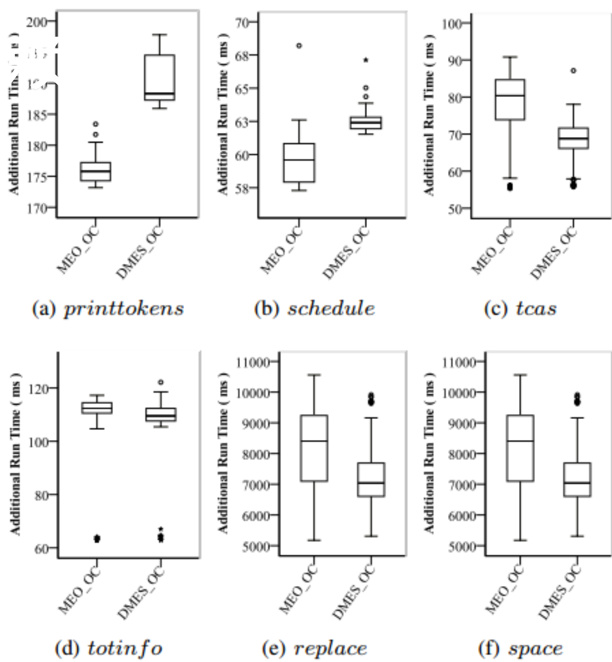
3 Empirical Evaluation

RQ3: How much additional run time is required by using the presented strategies?

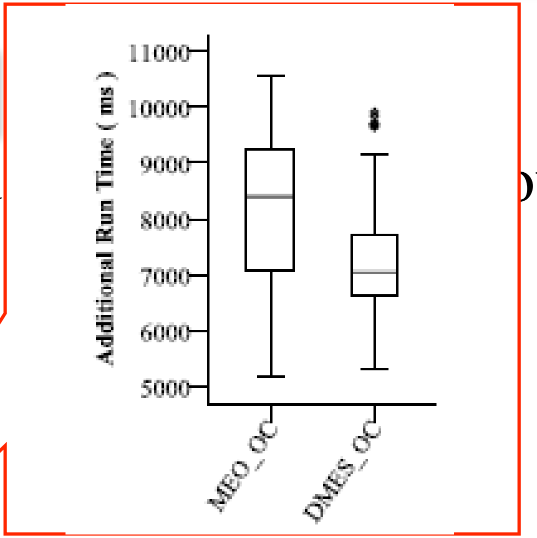
The actual run time of using DMES is used by excluding the mutation execution time.

Result and Analysis for RQ3

The additional run time is record[...]overhead of MEO and DMES.



Additional Run Time Cost of Faster-MBFL



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.

4 Conclusion

- 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.

The end

Q&A

Thank you for your attention!