An Investigation of Compression Techniques to Speed up Mutation Testing

Qianqian Zhu
Ph.D. student

Co-authors: Annibale Panichella and Andy Zaidman

Software Engineering Research Group,
Delft University of Technology, Netherlands

ICST 2018, April 12, 2017

Mutation Testing

An actively investigated field since 1970s

Mutation Testing

An actively investigated field since 1970s

- **Main idea:** small syntactic changes $\rightarrow$ test suite quality

Mutation Testing

An actively investigated field since 1970s

- **Main idea:** small syntactic changes $\rightarrow$ test suite quality
- **Benefit:**
  - better fault exposing capability
  - a good alternative to real faults

Mutation Testing

An actively investigated field since 1970s

- **Main idea:** small syntactic changes $\rightarrow$ test suite quality
- **Benefit:**
  - better fault exposing capability
  - a good alternative to real faults
- **limitations:**
  - high computational cost
  - undecidable Equivalent Mutant Problem

Mutation Testing

An actively investigated field since 1970s

- **Main idea:** small syntactic changes $\rightarrow$ test suite quality
- **Benefit:**
  - better fault exposing capability
  - a good alternative to real faults
- **limitations:**
  - high computational cost
  - undecidable Equivalent Mutant Problem

Our paper: speed up

Cost reduction techniques

Cost reduction techniques

- **Do fewer:** selecting fewer mutants;
  E.g., Selective Mutation

Offutt and Untch 2000

Cost reduction techniques

- **Do fewer:** selecting fewer mutants;

  E.g., Selective Mutation

- **Do smarter:** avoiding unnecessary test executions;

  E.g., Weak Mutation

Offutt and Untch 2000

Cost reduction techniques

- **Do fewer:** selecting fewer mutants;
  E.g., Selective Mutation
- **Do smarter:** avoiding unnecessary test executions;
  E.g., Weak Mutation
- **Do faster:** reducing the execution time;
  E.g., Mutation Schemata

Offutt and Untch 2000

Cost reduction techniques

- **Do fewer:** selecting fewer mutants;
  E.g., Selective Mutation
- **Do smarter:** avoiding unnecessary test executions;
  E.g., Weak Mutation
- **Do faster:** reducing the execution time;
  E.g., Mutation Schemata

Offutt and Untch 2000

Cost reduction techniques

- **Do fewer:** selecting fewer mutants;
  E.g., Selective Mutation
- **Do smarter:** avoiding unnecessary test executions;
  E.g., Weak Mutation
- **Do faster:** reducing the execution time;
  E.g., Mutation Schemata

Challenged by Gopinath et al. (ICSE 2016,TR 2017):
No practical advantage over **pure random sampling**

Offutt and Untch 2000

Mutation data compression

Mutation data compression

647 KB

Mutation data compression

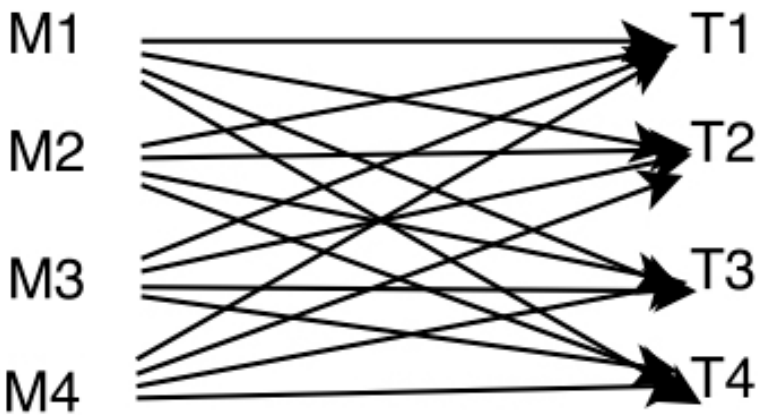
647 KB

19 KB

Mutation data compression

647 KB

19 KB

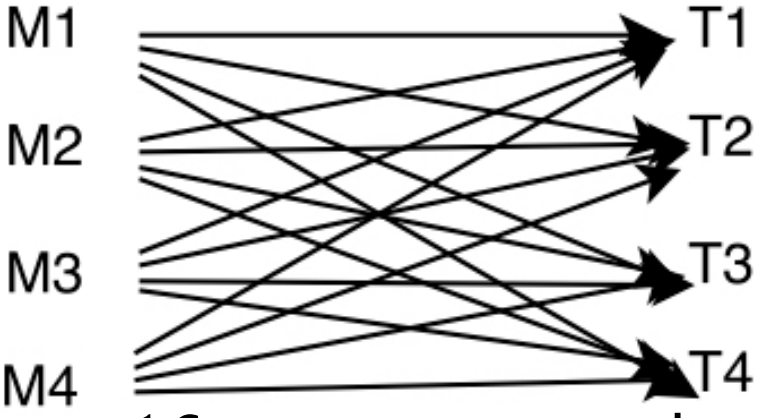


16 tests executed

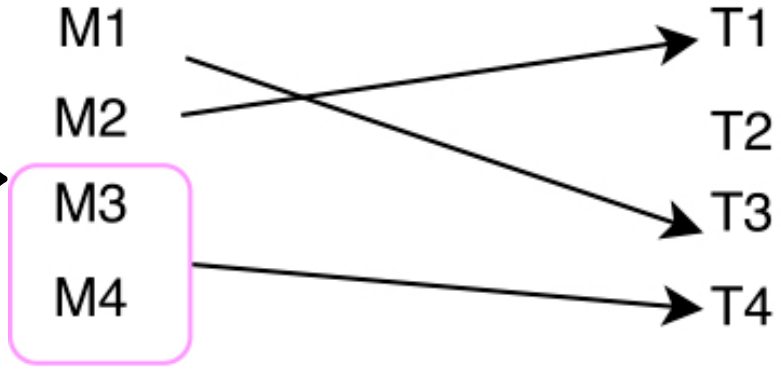
Mutation data compression

647 KB

19 KB



16 tests executed



3 tests executed

Weak Mutation

For a test case t to kill a mutant m which mutates the statement s of a program P, there are **three conditions:**

Weak Mutation

For a test case t to kill a mutant m which mutates the statement s of a program P, there are **three conditions:**

- Reachability
- Necessity
- Sufficiency

Weak Mutation

For a test case t to kill a mutant m which mutates the statement s of a program P, there are **three conditions:**

- Reachability
- Necessity
- Sufficiency

```
    Program:
1   int fun(int a, int b){
2     int c;
3     if(a>0)
4       c=a+b;
5     else
6       c=a-b;
7     return abs(c);
8   }
```

Test:
assertEquals(fun(0,-1),1)

Weak Mutation

For a test case t to kill a mutant m which mutates the statement s of a program P, there are **three conditions:**

- Reachability
- Necessity
- Sufficiency

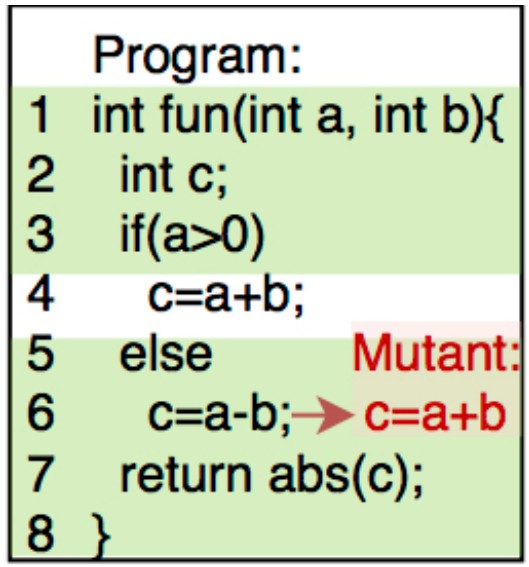
```
   Program:
1  int fun(int a, int b){
2    int c;
3    if(a>0)
4      c=a+b;
5    else          Mutant:
6      c=a-b;  →   c=a+b
7    return abs(c);
8  }
```

```
Test:
assertEquals(fun(0,-1),1)
```

Weak Mutation

For a test case t to kill a mutant m which mutates the statement s of a program P, there are **three conditions:**

- Reachability $\rightarrow$ statement coverage
- Necessity
- Sufficiency



Test:
assertEquals(fun(0,-1),1)

Weak Mutation

For a test case t to kill a mutant m which mutates the statement s of a program P, there are **three conditions:**

- Reachability $\rightarrow$ statement coverage
- Necessity $\rightarrow$ weak mutation
- Sufficiency

```
   Program:
1  int fun(int a, int b){
2    int c;
3    if(a>0)
4      c=a+b;
5    else          Mutant:
6      c=a-b;  →   c=a+b
7    return abs(c);
8  }
```

Test:
assertEquals(fun(0,-1),1)

Weak Mutation

For a test case t to kill a mutant m which mutates the statement s of a program P, there are **three conditions:**

- Reachability → statement coverage
- Necessity → weak mutation
- Sufficiency → strong mutation

```
   Program:
1  int fun(int a, int b){
2    int c;
3    if(a>0)
4      c=a+b;
5    else          Mutant:
6      c=a-b;  →   c=a+b
7    return abs(c);
8  }
```

Test:
assertEquals(fun(0,-1),1)

Weak Mutation

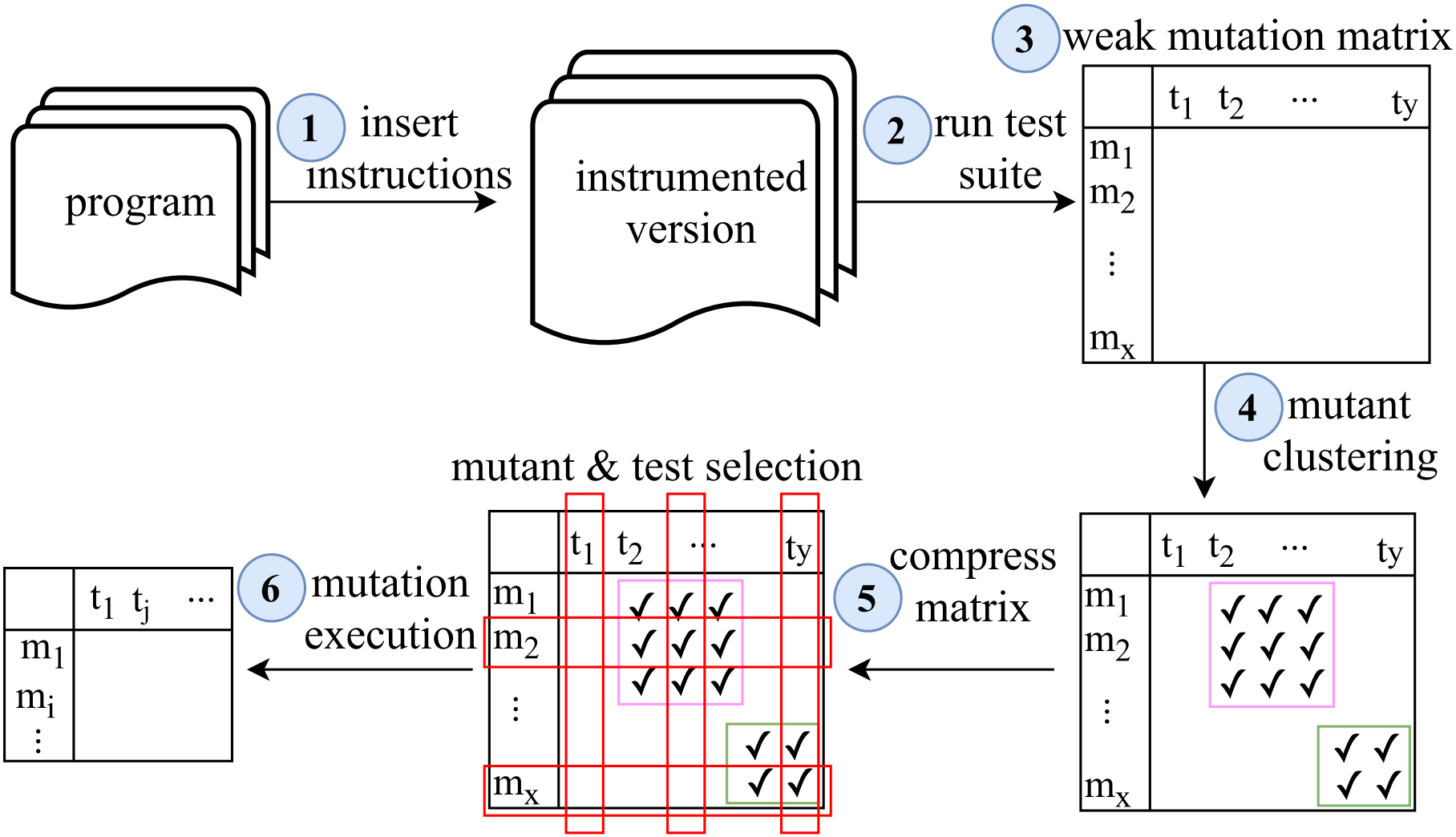
For a test case t to kill a mutant m which mutates the statement s of a program P, there are **three conditions:**

- Reachability → statement coverage
- Necessity → weak mutation
- Sufficiency → strong mutation

```
   Program:
1  int fun(int a, int b){
2    int c;
3    if(a>0)
4      c=a+b;
5    else          Mutant:
6      c=a-b;  →   c=a+b
7    return abs(c);
8  }
```

Test:
assertEquals(fun(0,-1),1)

Overall methodology



Mutant clustering

Based on weak mutation

- **Overlapped grouping**
  elements are only grouped together if they are identical

- **FCA-based grouping**
  Formal Concept Analysis; convey binary relations

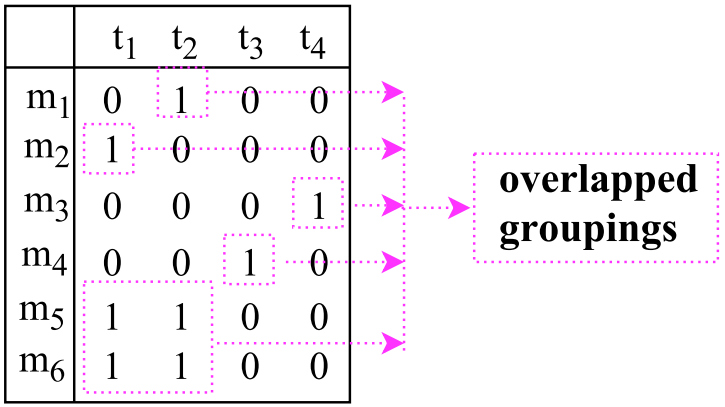
	t_1	t_2	t_3	t_4
m_1	0	1	0	0
m_2	1	0	0	0
m_3	0	0	0	1
m_4	0	0	1	0
m_5	1	1	0	0
m_6	1	1	0	0

Mutant clustering

Based on weak mutation

- **Overlapped grouping**
  elements are only grouped together if they are identical

- **FCA-based grouping**
  Formal Concept Analysis; convey binary relations

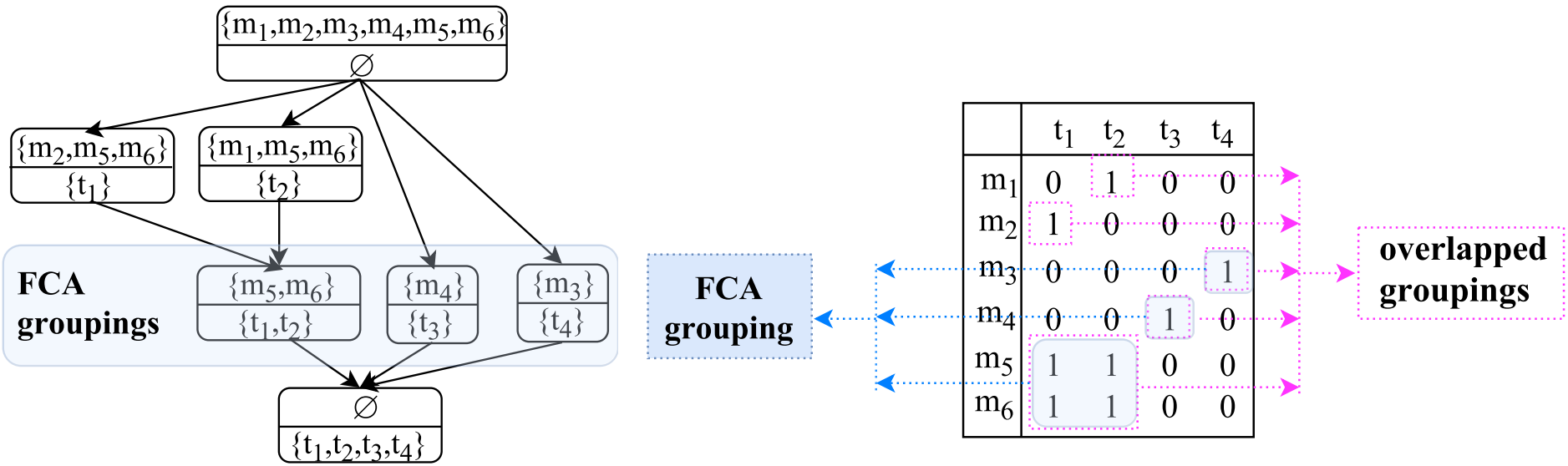


Mutant clustering

Based on weak mutation

- **Overlapped grouping**
  elements are only grouped together if they are identical

- **FCA-based grouping**
  Formal Concept Analysis; convey binary relations



Mutant selection strategies

- **Cluster-based selection (CS)**

  randomly chooses one mutant from each cluster
- **CS + mutation operator type (mop)**

  divide each cluster into partitions by mutation operator types and then random selection
- **CS + mutation location (mloc)**

  divide each cluster into partitions by mutant locations and then random selection

	mop	mloc
m_1	1	Line5
m_2	1	Line5
m_3	2	Line5
m_4	2	Line6

Mutant selection strategies

- **Cluster-based selection (CS)**

  randomly chooses one mutant from each cluster
- **CS + mutation operator type (mop)**

  divide each cluster into partitions by mutation operator types and then random selection
- **CS + mutation location (mloc)**

  divide each cluster into partitions by mutant locations and then random selection

	mop	mloc
m_1	1	Line5
m_2	1	Line5
m_3	2	Line5
m_4	2	Line6

Mutant selection strategies

- **Cluster-based selection (CS)**

  randomly chooses one mutant from each cluster
- **CS + mutation operator type (mop)**

  divide each cluster into partitions by mutation operator types and then random selection
- **CS + mutation location (mloc)**

  divide each cluster into partitions by mutant locations and then random selection

	mop	mloc
m_1	1	Line5
m_2	1	Line5
m_3	2	Line5
m_4	2	Line6

Mutant selection strategies

- **Cluster-based selection (CS)**

  randomly chooses one mutant from each cluster
- **CS + mutation operator type (mop)**

  divide each cluster into partitions by mutation operator types and then random selection
- **CS + mutation location (mloc)**

  divide each cluster into partitions by mutant locations and then random selection

	mop	mloc
m_1	1	Line5
m_2	1	Line5
m_3	2	Line5
m_4	2	Line6

Empirical study

- **20 open-source projects (Java):** 397K+ LOC, 2K+ tests, 166K+ mutants
- **6+2 methods:**
  - overlap
  - overlap+mop
  - overlap+mloc
  - fca
  - fca+mop
  - fca+mloc
  - pure random
  - weak mutation
- **Research questions:**
  - **RQ1:** How accurate are different compression techniques?
  - **RQ2:** How do compression techniques perform in terms of speed-up?
  - **RQ3:** What is the trade-off between accuracy and speed-up?

Empirical study

- **20 open-source projects (Java):** 397K+ LOC, 2K+ tests, 166K+ mutants
- **6+2 methods:**
  - overlap
  - overlap+mop
  - overlap+mloc
  - fca
  - fca+mop
  - fca+mloc
  - pure random
  - weak mutation

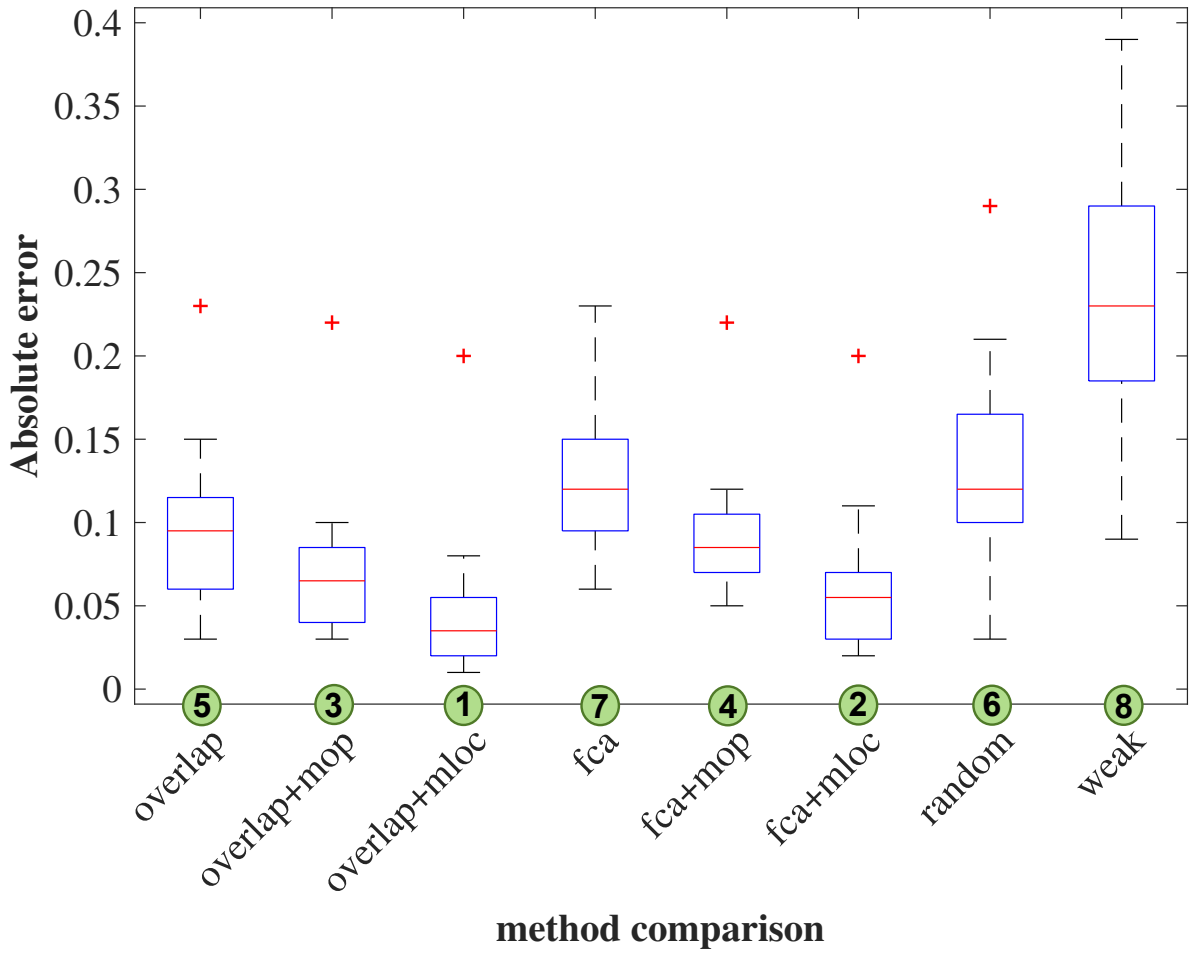
  ↓ baselines
- **Research questions:**
  - **RQ1:** How accurate are different compression techniques?
  - **RQ2:** How do compression techniques perform in terms of speed-up?
  - **RQ3:** What is the trade-off between accuracy and speed-up?

RQ1: accuracy performance

- **Absolute error**

$$AE(C,T) = | \, strong_M(C,T) - estimated_M(C,T) \, | \qquad (1)$$

- **Results:**

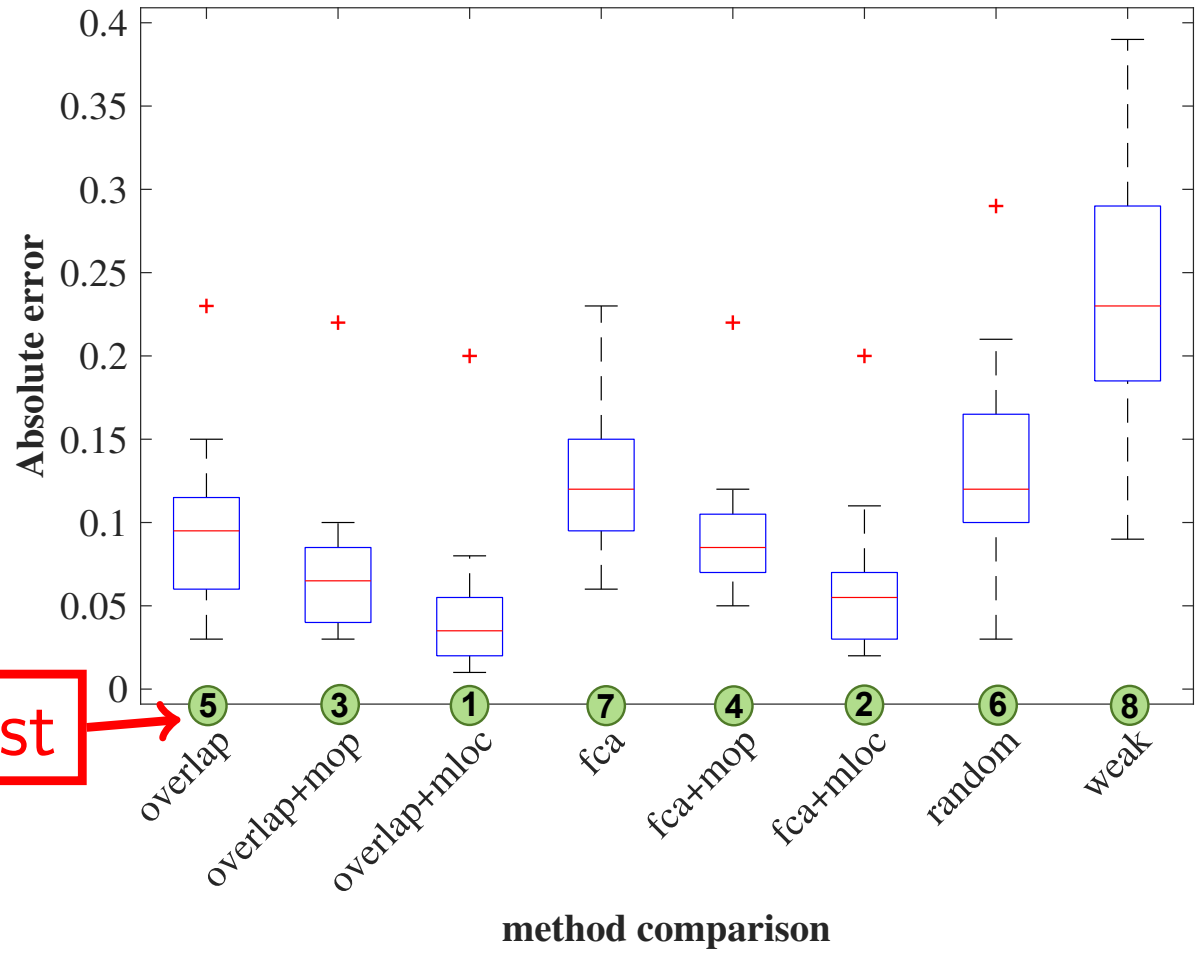


RQ1: accuracy performance

- **Absolute error**

$$AE(C,T) = | \ strong_M(C,T) - estimated_M(C,T) \ | \qquad (1)$$

- **Results:**



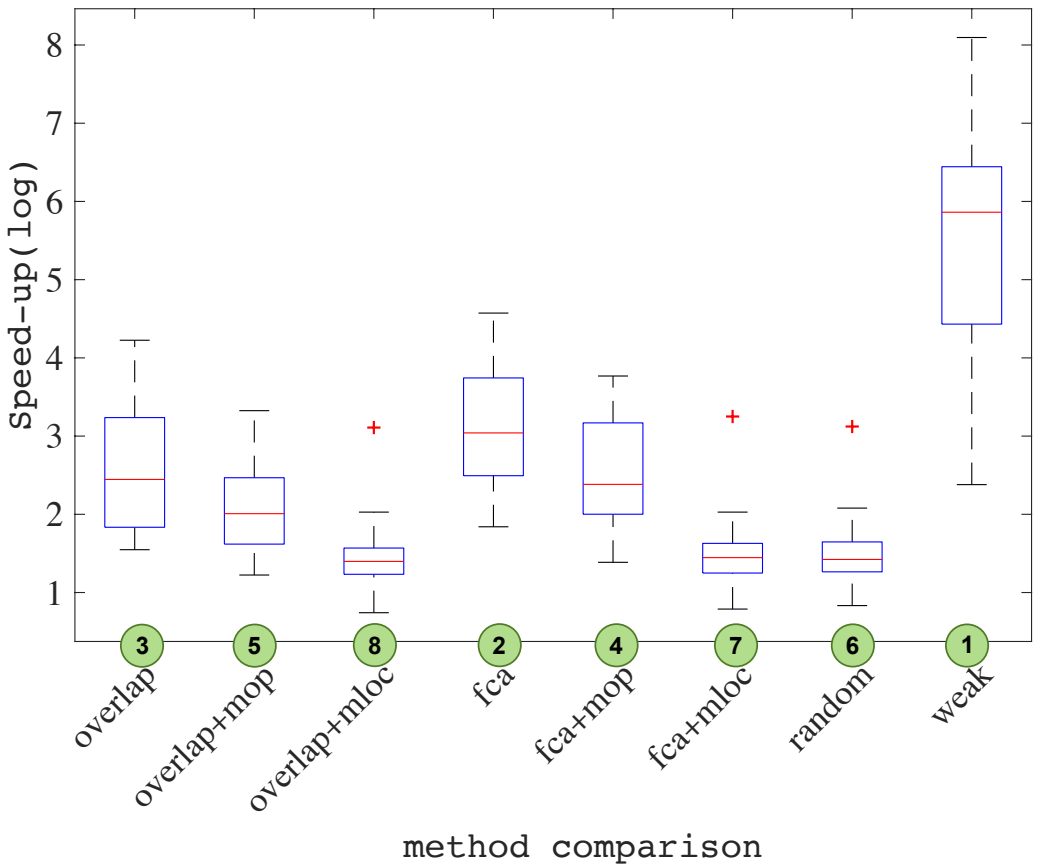
Friedman's test

RQ2: Speed-up performance

- **Speed-up**

$$speed\text{-}up = \frac{exec\_time(strong\_mutation)}{exec\_time(approach)} \quad (2)$$

- **Results:**

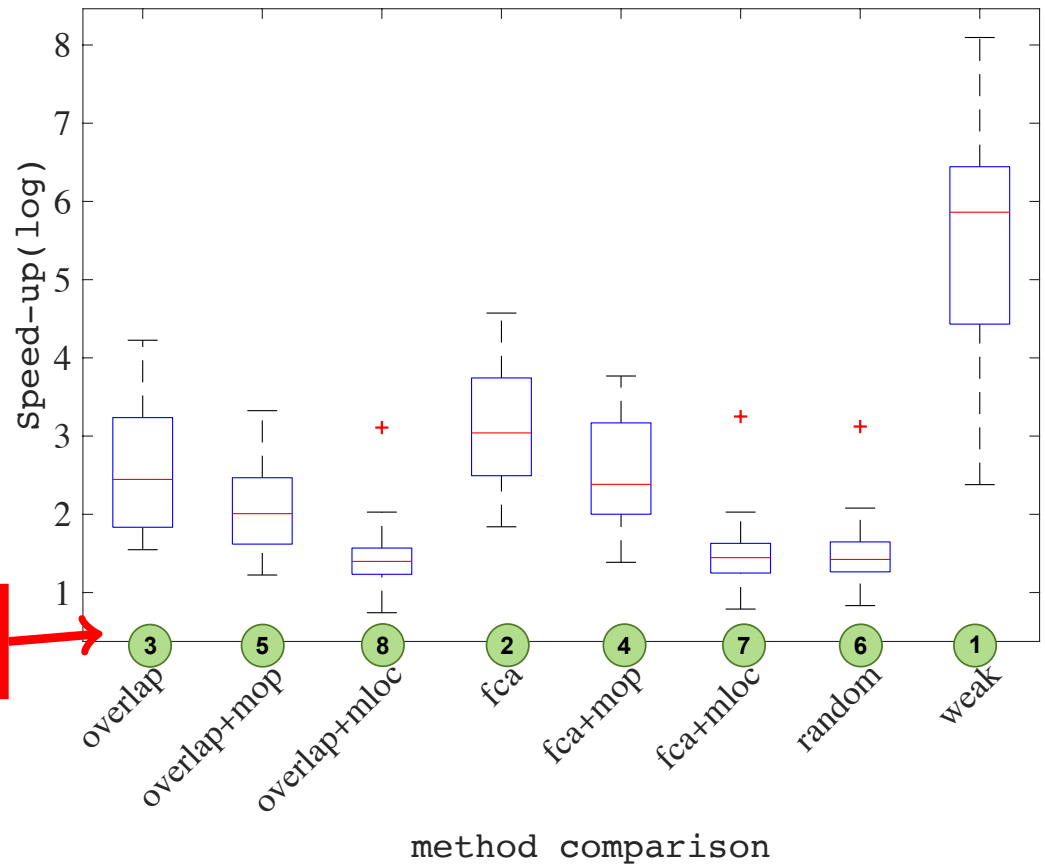


RQ2: Speed-up performance

- **Speed-up**

$$speed\text{-}up = \frac{exec\_time(strong\_mutation)}{exec\_time(approach)} \quad (2)$$

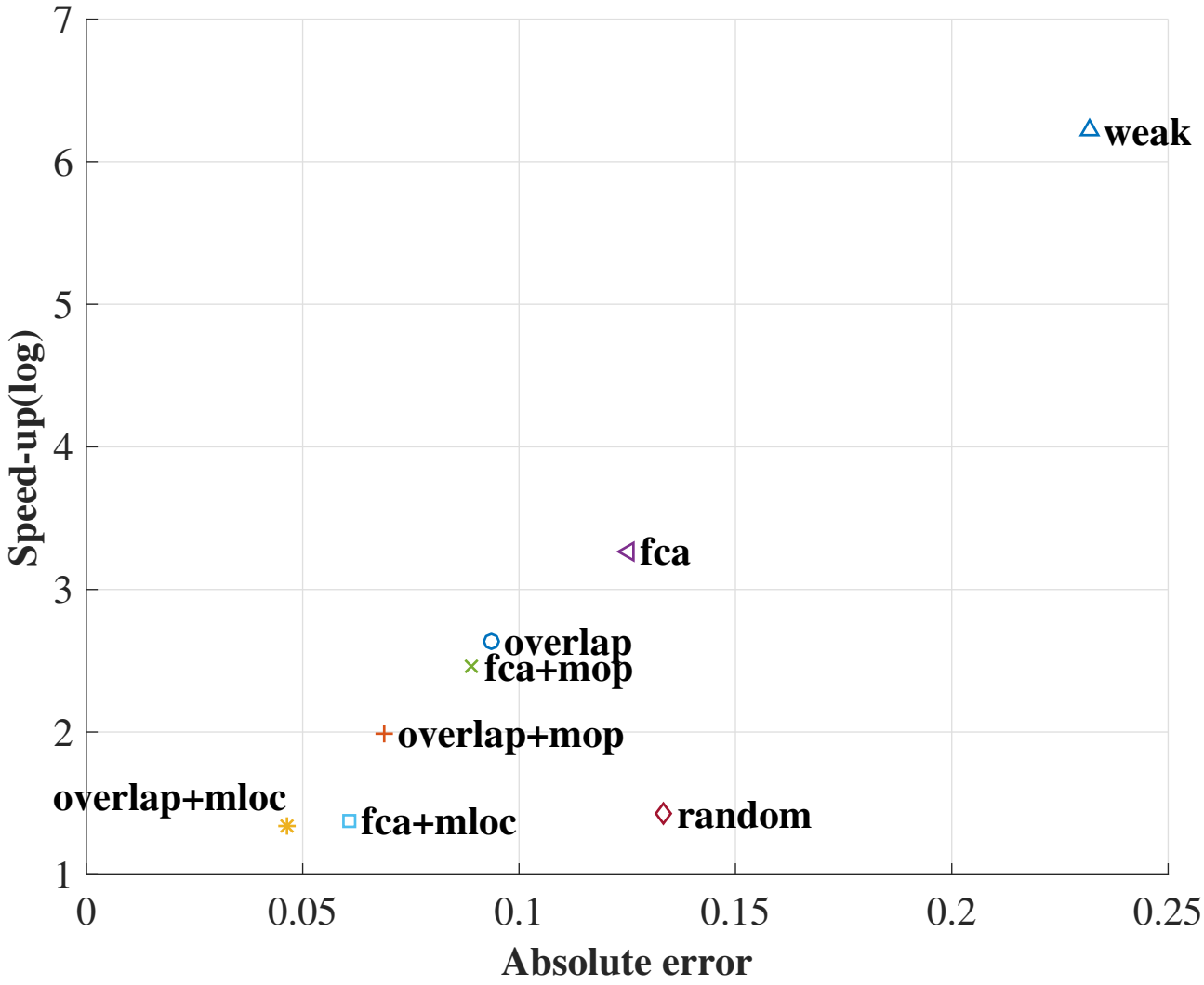
- **Results:**



Friedman's test

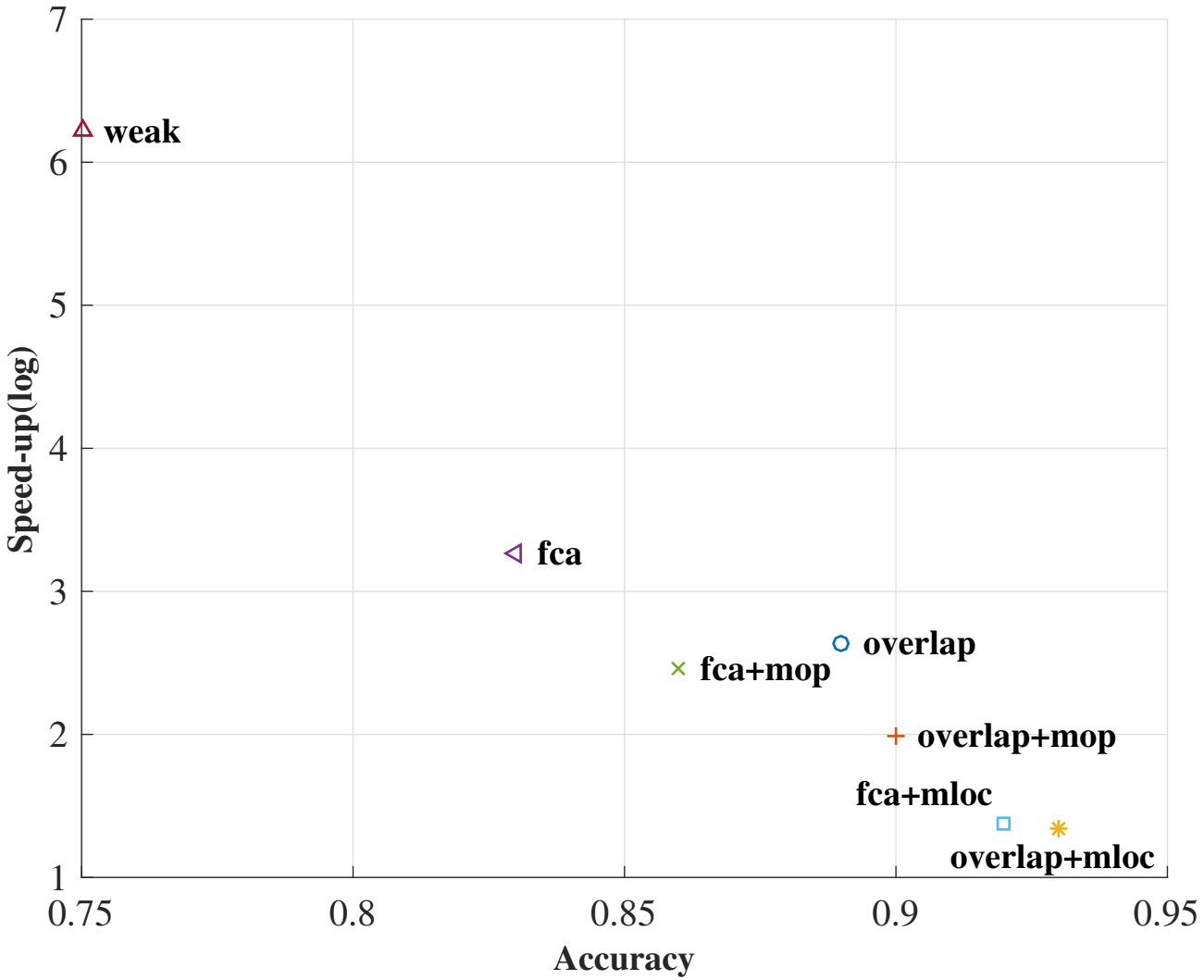
RQ3: Trade-off

Speed-up v.s. absolute error (mean)



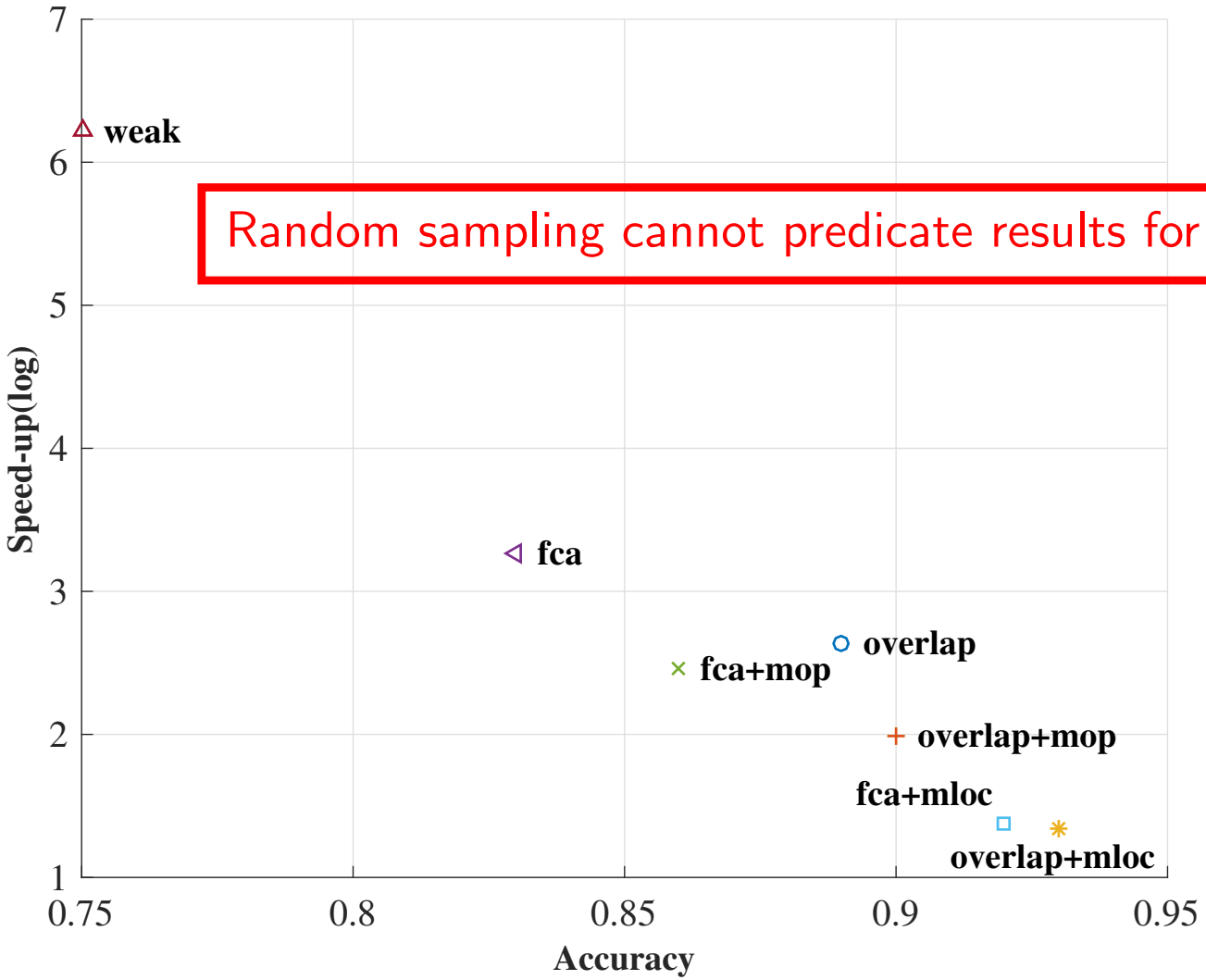
RQ3: Trade-off

Speed-up v.s. prediction accuracy (mean)



RQ3: Trade-off

Speed-up v.s. prediction accuracy (mean)



Sum-up

Sum-up

- Mutation compression v.s. random sampling:

Sum-up

- Mutation compression v.s. random sampling:
  - Mutation compression: speed-up $\uparrow$ and accuracy $\uparrow$
  - Killable mutant results outperform overall mutation score

Sum-up

- Mutation compression v.s. random sampling:
  - Mutation compression: speed-up $\uparrow$ and accuracy $\uparrow$
  - Killable mutant results outperform overall mutation score
- Mutation location trumps mutation operator

Sum-up

- Mutation compression v.s. random sampling:
  - Mutation compression: speed-up $\uparrow$ and accuracy $\uparrow$
  - Killable mutant results outperform overall mutation score
- Mutation location trumps mutation operator
- Future work:
  - Combining mutation operator and mutation location
  - Exploring other compression techniques
  - Applying to test-data generation