An Empirical Study of Fault Localization Families and Their Combinations

Daming Zou, Jingjing Liang, Yingfei Xiong, Michael D. Ernst, and Lu Zhang

ESEC/FSE 2019

Journal First Paper on TSE

Tallinn, Estonia

29 Aug 2019

PEKING UNIVERSITY 1898

UNIVERSITY of WASHINGTON

Fault Localization (FL)

- Automated Fault Localization
  - Using static and run-time information to locate the root cause of failure.
  - E.g., test coverage, program dependency, test output, etc.

  - Typical output, a ranked suspicious list:

```
foo.java, line 12
foo.java, line 10   (Bingo!)
bar.java, line 5
...
```

Fault Localization Families

FL Family	Information Source
Spectrum-based (SBFL)	Test coverage information
Mutation-based (MBFL)	Info from mutating the program
(Dynamic) Slicing	Dynamic program dependencies
Stack trace analysis	Stack trace when crash
Predicate switching	Info from mutating the results of conditional expressions
Information retrieval-based (IR-based)	Bug reports
History-based	Development history

Motivation

- Existing studies focus on comparison within family:

Ochiai(SBFL) vs. DStar(SBFL) vs. Tarantula(SBFL) vs. …

- This study tries to understand the correlation of different families on real-world dataset. In terms of both effectiveness and efficiency.

	Performance	Run-time cost
SBFL	?	?
MBFL	?	?
etc.	?	?

This empirical study…

- Covered a wide range of FL techniques from 7 families.
- Based on 357 real-world faults from Defects4j dataset.
- Proposed a combined technique that significantly outperforms all existing techniques.

Research Questions

- RQ1: How effective are the standalone FL techniques?
- RQ2: How much are these techniques correlated?
  - Reveals the possibility of combining them.
- RQ3: How effectively can we combine these techniques?
- RQ4: What is the run-time cost of standalone and combined techniques?

Experimental Subjects

- Defects4j dataset
- 5 real-world and widely-used projects.
- 357 actual faults.
- Average size of projects: 138,000 lines of code.

Project	Faults	LoC
Apache Commons **Math**	106	103.9k
Apache Commons **Lang**	65	49.9k
Joda-**Time**	27	105.2k
JFree**Chart**	26	132.2k
Google **Closure** compiler	133	216.2k
Total	357	138.0k

RQ1. Effectiveness of Standalone Techniques

- Top *n*: How many faults can be localized within top n positions.
- The effectiveness differs significantly between families.
- Spectrum-based FL is the most effective family.

TABLE 3
The Performance of Standalone Techniques on all 357 faults. Boldface indicates the best-performing techniques.

Family	Technique	$E_{inspect}$ @1	@3	@5	@10	*EXAM*
SBFL	Ochiai	16 (4%)	81 (23%)	**111 (31%)**	**156 (44%)**	**0.033**
	DStar	17 (5%)	**84 (24%)**	**111 (31%)**	155 (43%)	**0.033**
MBFL	Metallaxis	23 (6%)	78 (22%)	103 (29%)	129 (36%)	0.118
	MUSE	**24 (7%)**	44 (12%)	58 (16%)	68 (19%)	0.304
slicing	union	5 (1%)	33 (9%)	58 (16%)	84 (24%)	0.207
	intersection	5 (1%)	35 (10%)	55 (15%)	71 (20%)	0.222
	frequency	6 (2%)	39 (11%)	58 (16%)	84 (24%)	0.208
stack trace	stack trace	20 (6%)	31 (9%)	38 (11%)	38 (11%)	0.311
predicate switching	predicate switching	3 (1%)	15 (4%)	20 (6%)	23 (6%)	0.331
IR-based	BugLocator	0 (0%)	0 (0%)	0 (0%)	3 (1%)	0.212
history-based	Bugspots	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0.465

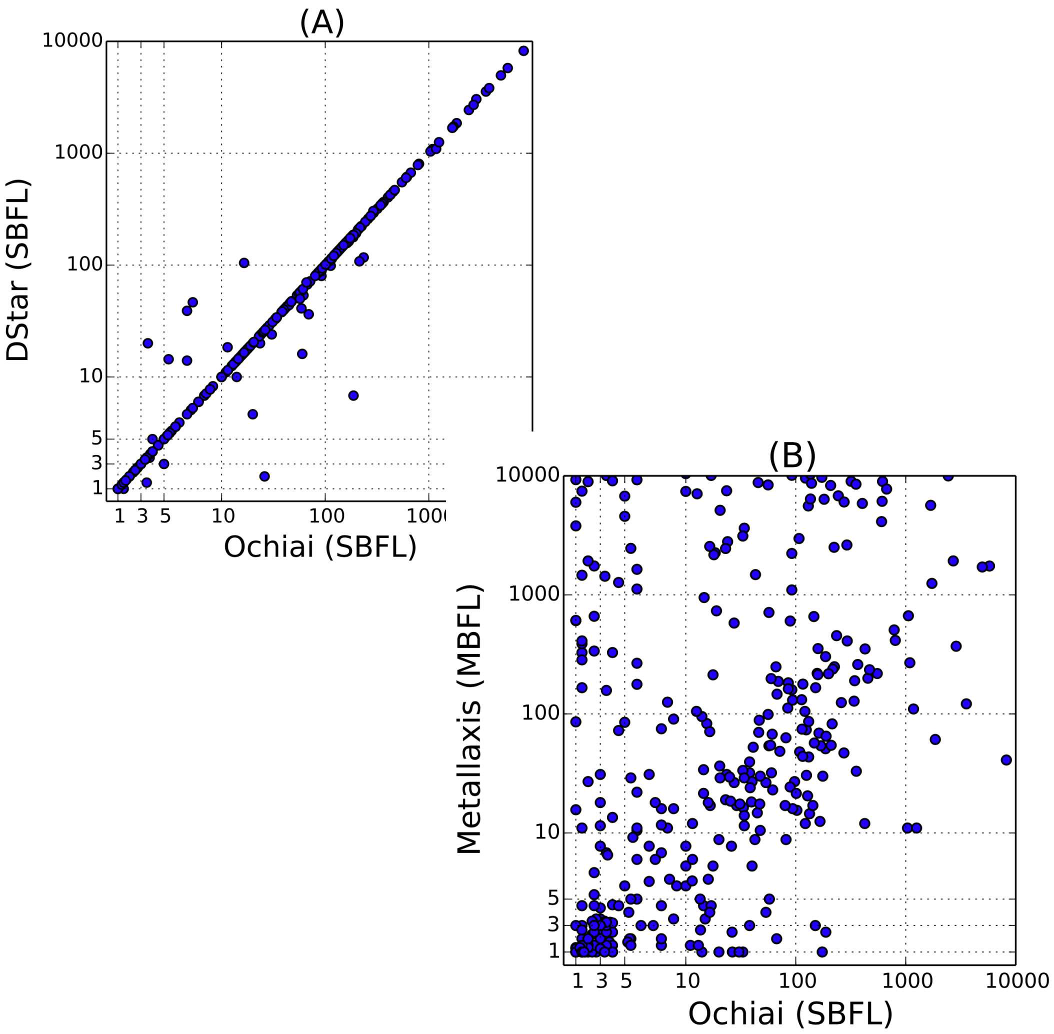
RQ1. Effectiveness of Standalone Techniques

- Stack trace analysis is the most effective one on *crash faults*.

TABLE 4
The Performance of Techniques on *Crash Faults* (90 out of 357 faults, 25%)

Family	Technique	$E_{inspect}$ @1	@3	@5	@10	*EXAM*
SBFL	Ochiai	4 (4%)	17 (19%)	32 (36%)	**50 (56%)**	**0.028**
	DStar	4 (4%)	18 (20%)	33 (37%)	**50 (56%)**	0.029
MBFL	Metallaxis	10 (11%)	30 (33%)	35 (39%)	44 (49%)	0.083
	MUSE	6 (7%)	13 (14%)	18 (20%)	19 (21%)	0.345
slicing	union	2 (2%)	13 (14%)	26 (29%)	36 (40%)	0.112
	intersection	2 (2%)	13 (14%)	21 (23%)	30 (33%)	0.136
	frequency	2 (2%)	14 (16%)	25 (28%)	36 (40%)	0.112
stack trace	stack trace	**20 (22%)**	**31 (34%)**	**38 (42%)**	38 (42%)	0.194
predicate switching	predicate switching	1 (1%)	5 (6%)	8 (9%)	9 (10%)	0.323
IR-based	BugLocator	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0.199
history-based	Bugspots	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0.433

RQ2. Correlation between Techniques



- 55 pairs of techniques in total.
- Only 2 pairs are significantly correlated.
  - Ochiai(SBFL) / Dstar(SBFL)
  - Union(Slicing) / Frequency(Slicing)

- Most techniques are weakly correlated, including all techniques in different families.

- Possibility to utilize the potential complementary information.

RQ3. Effectiveness of Combining Techniques

- How to combine? Learning to Rank.
  - First introduced to FL by Xuan & Monperrus[1].
  - Standalone techniques are treated as a black box.
  - Output: One re-ranked suspicious list.

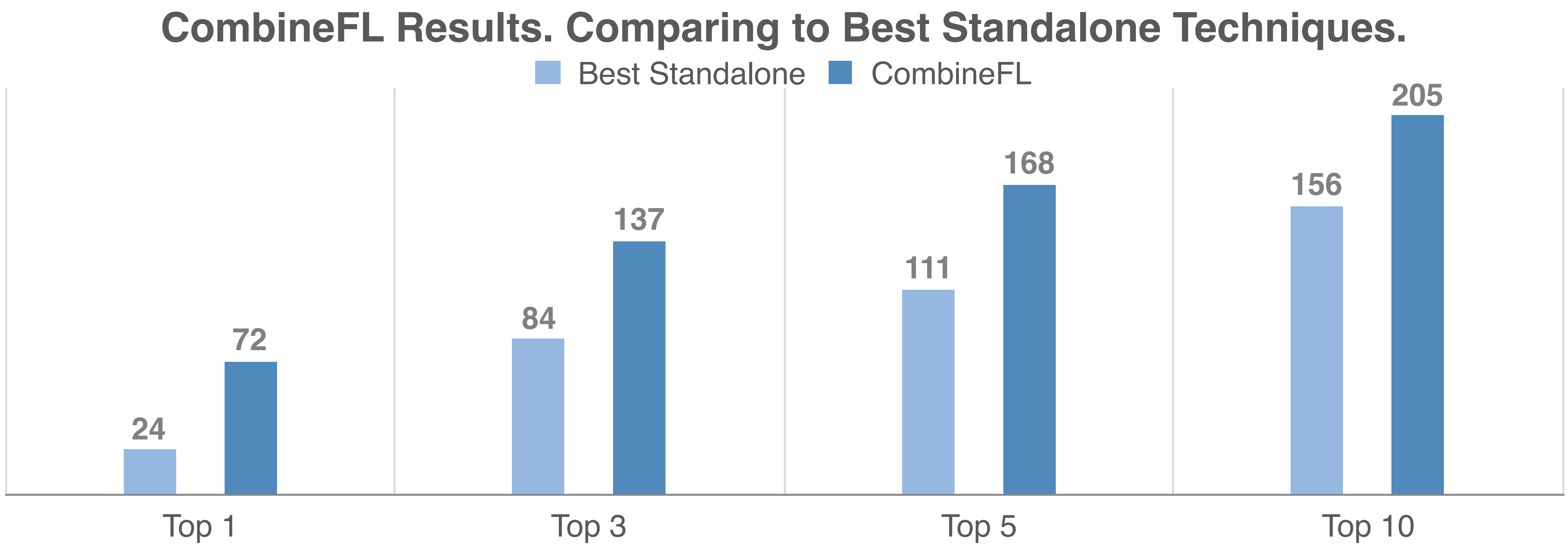
- Example:

```
foo.java line 12: {Ochiai: 0.6, slicing: 0, MUSE: 0.3, …}
foo.java line 10: {Ochiai: 0.5, slicing: 1, MUSE: 0.3, …}
bar.java line 5:  {Ochiai: 0.4, slicing: 1, MUSE: 0.4, …}
```

[1] Xuan, Jifeng, and Martin Monperrus. "Learning to combine multiple ranking metrics for fault localization." *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014.

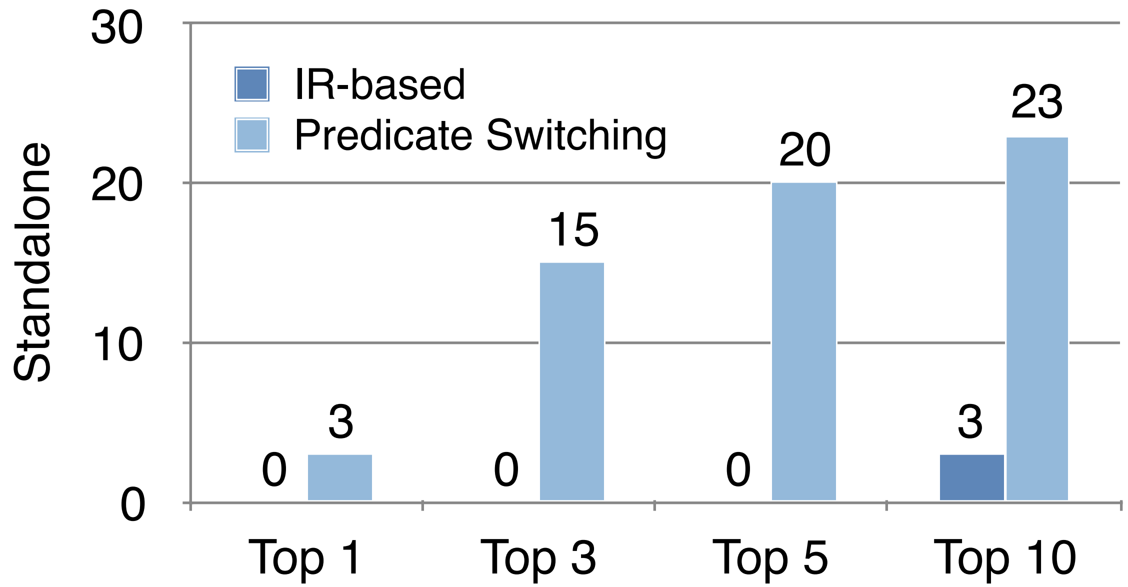
RQ3. Effectiveness of Combining Techniques

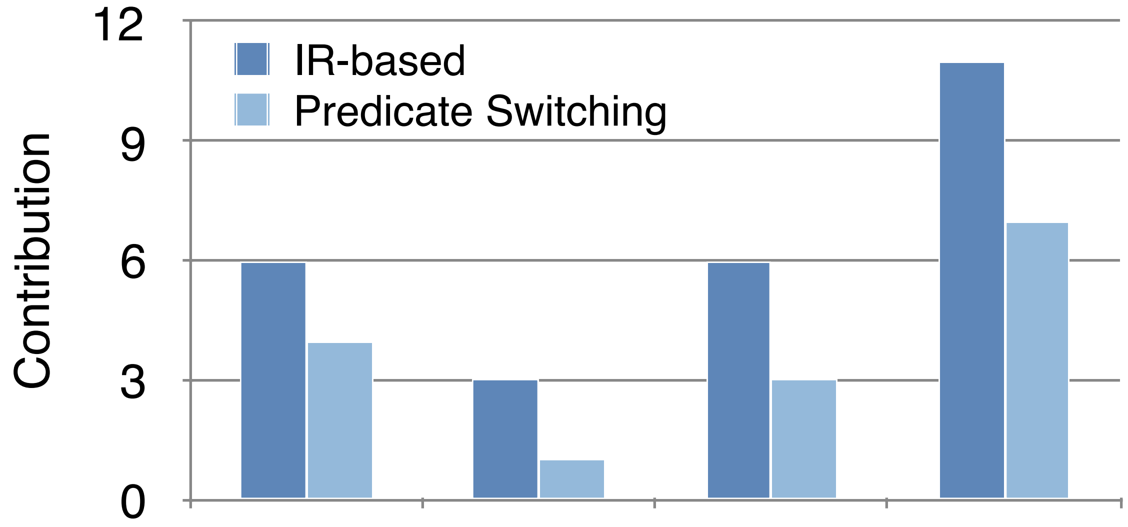
- The combined technique significantly outperforms any standalone technique.



RQ3. Effectiveness of Combining Techniques

- Contribution: decrease when remove from the combination.
- The contribution of each technique to the combined results is not determined by its effectiveness as a standalone technique.





RQ4. Time Consumption and Combination Strategy

- FL families can be categorized into levels.
- The run-time differs in orders of magnitude between levels.

(in seconds)

Time Level	Family	Technique	Average
Level 1 (Seconds)	history-based	Bugspots	0.54
	stack trace	stack trace	1.3
	IR-based	BugLocator	5.6
Level 2 (Minutes)	slicing	union intersection frequency	80 80 80
	SBFL	Ochiai DStar	200 200
Level 3 (Around ten minutes)	predicate switching	predicate switching	620
Level 4 (Hours)	MBFL	Metallaxis MUSE	4800 4800

RQ4. Time Consumption and Combination Strategy

- How to select FL techniques for combination:
  - Select an acceptable time level.
  - Include all preceding level families.

Time Level	Technique	Estimated Time (in seconds)	$E_{inspect}$ @1	@3	@5	@10	*EXAM*
Level 1	history-based	0.54	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0.465
	stack trace	1.3	19 (5%)	29 (8%)	35 (10%)	35 (10%)	0.311
	stack trace +history-based	13	19 (5%)	29 (8%)	35 (10%)	35 (10%)	0.311
	stack trace +history-based +IR-based	19	**25 (7%)**	**42 (12%)**	**53 (15%)**	**63 (18%)**	**0.0421**
Level 2	Level 1 +slicing	98	28 (8%)	65 (18%)	95 (27%)	124 (35%)	0.0353
	Level 1 +SBFL	220	39 (11%)	105 (29%)	132 (37%)	174 (49%)	0.0244
	Level 1 +SBFL +slicing	300	**52 (15%)**	**120 (34%)**	**146 (41%)**	**189 (53%)**	**0.0217**
Level 3	Level 2 +predicate switching	920	52 (15%)	122 (34%)	148 (41%)	194 (54%)	0.0206
Level 4	Level 3 +MBFL	5700	72 (20%)	137 (38%)	168 (47%)	205 (57%)	0.0173

Implications

- Call for more information sources.
- Evaluating a FL technique:
  - It is important to know its contribution to the existing combinations.
- Both effectiveness and efficiency are important.

- Our infrastructure available at:

  https://combinefl.github.io/

  - Standard JSON format.
  - Automated integrating your FL technique with all aforementioned techniques.