A Bayesian Framework for Automated Debugging

[Sungmin Kang, Wonkeun Choi], Shin Yoo
Presented on 2023-07-18 by Sungmin
Painting by Georges Seurat, *The Channel at Gravelines*, 1890

KAIST
COINSE
ISSTA 2023

1.
Automated Debugging

Usual Automated Debugging

Failing Test | **Fault Localization** | **Program Repair (APR)**

Using a failing test, among other info...

Finding which file, function, line actually contains bug

Correcting the code

All want to use available information efficiently

Failing Test

Using a failing test, among other info...

Fault Localization

SBFL: Using Coverage Information
MBFL: Using Mutation Info
IRFL: Using Lexical Affinity to Bug Report...

Program Repair (APR)

Template-based: apply appropriate patches using syntax information
Neural: neural networks using syntax information
...

…but no theory to analyze/provide directions

Benefits of having a theoretic framework

Clarification of assumptions

Concrete suggestions from predictions

Curation of new research ideas

Presentation Organization

Clarification of assumptions

Concrete suggestions from predictions

Curation of new research ideas

2. Bayesian Framework

3. Prioritizing Patches

4. Future Work

2.
Bayesian Framework

Bayesian Inference

Bayesian inference is a method of statistical inference in which Bayes' theorem is used to update the probability for a hypothesis as more evidence or information becomes available. Bayesian inference is an important technique in statistics, and especially in mathematical statistics. Bayesian updating is particularly

$$P(H \mid E) = \frac{P(E \mid H) \cdot P(H)}{P(E)}$$

(From Wikipedia)

Automated Debugging Plugins

$$P(H \mid E) = \frac{P(E \mid H) \cdot P(H)}{P(E)}$$

- As hypotheses (H), we set debugging results, e.g. “line k is buggy”.
- As evidence (E), we set execution results, e.g. “test t failed”.

The “posterior”

$$P(H \mid E) = \frac{P(E \mid H) \cdot P(H)}{P(E)}$$

- What we ultimately want to know is P(H|E): How likely is it that hypothesis H is true, given evidence E?
    - For example, “How likely is it that line k is the buggy line, given test t failed?
- However, this term is difficult to calculate directly. Thus we calculate it using the terms on the right-hand side…

Right-hand-side terms

$$P(H \mid E) = \frac{P(E \mid H) \cdot P(H)}{P(E)}$$

- P(E|H) [likelihood]: Assuming the hypothesis, how likely is the evidence?
  - For example, “If line k is the buggy line, how likely is that test t would fail?”
- P(H) [prior]: Prior to seeing the evidence E, how likely was the hypothesis?
  - For example, “How likely was line k to be the buggy line the first place?”
- P(E): A normalization term (not important for our purposes)

Overall, in automated debugging:

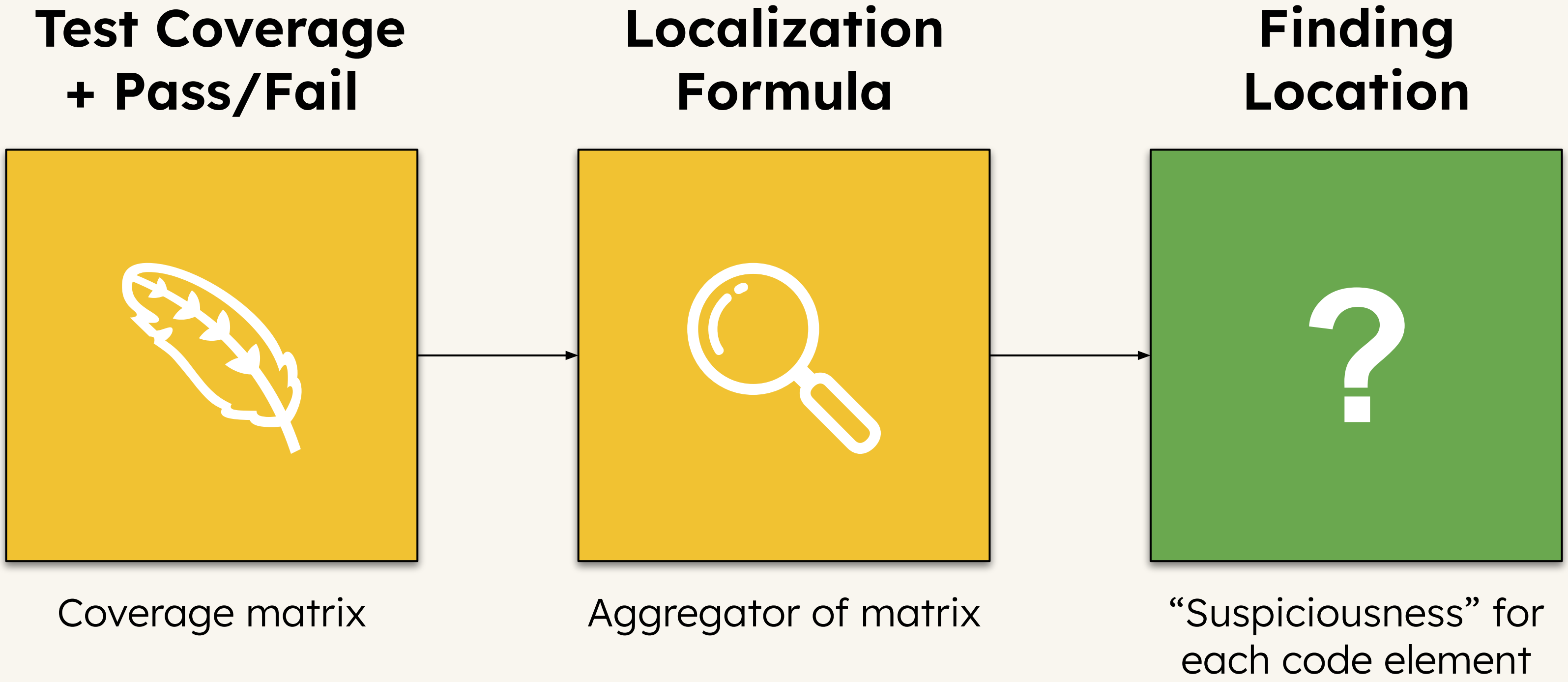
- We want to infer the location (l) to apply a fix action (a), based on the available data (D):

$$P(l, a|D) \propto P(D|l, a)P(l, a)$$

- And fault localization is a special case of automated debugging, with probabilities marginalized over the action space:

$$P(l|D) \propto P(D|l)P(l)$$

First example of analysis: SBFL



Many formulae have been proposed

Table 2: Risk Evaluation formulæ

Name	Formula	Name	Formula
Jaccard [4]	$\frac{e_f}{e_f+n_f+e_p}$	Ochiai [5]	$\frac{e_f}{\sqrt{(e_f+n_f)\cdot(e_f+e_p)}}$
Tarantula [7]	$\frac{\frac{e_f}{e_f+n_f}}{\frac{e_p}{e_p+n_p}+\frac{e_f}{e_f+n_f}}$	AMPLE [6]	$\left\|\frac{e_f}{e_f+n_f} - \frac{e_p}{e_p+n_p}\right\|$
Wong1 [9]	e_f	Wong2 [9]	$e_f - e_p$
Wong3 [9]	$e_f - h$, where $h = \begin{cases} e_p & \text{if } e_p \leq 2 \\ 2 + 0.1(e_p - 2) & \text{if } 2 < e_p \leq 10 \\ 2.8 + 0.001(e_p - 10) & \text{if } e_p > 10 \end{cases}$		
Op1 [8]	$\begin{cases} -1 & \text{if } n_f > 0 \\ n_p & \text{otherwise} \end{cases}$	Op2 [8]	$e_f - \frac{e_p}{e_p+n_p+1}$

(From Yoo, *Evolving Human Competitive Spectra-Based Fault Localisation Techniques*, 2013)

Assumptions about SBFL

1. **Single Fault Assumption** - assumes a single fault, for simplicity.

2. **Deterministic Code** - no flakiness.

3. **Existence of failing test case** - without one, it would be impossible to locate a bug.

Likelihood Function

- If a test executes the fault, the test will fail with probability p
- If a test doesn't execute the fault, it cannot fail

$$P(t = \text{fail} | l = \text{fault} \land l \in_c t) = p$$

$$P(t = \text{fail} | l = \text{fault} \land l \notin_c t) = 0$$

Simplified Conditional Probability

$$P(l = \text{fault} | t_1, \dots, t_n) \propto \begin{cases} 0 & e_f < F \\ (1-p)^{e_p} & e_f = F \end{cases}$$

Which is equivalent to Op1, one of the maximal formulae from Yoo et al:

Op1 [8]	$\begin{cases} -1 & \text{if } n_f > 0 \\ n_p & \text{otherwise} \end{cases}$	Op2 [8]	$e_f - \frac{e_p}{e_p + n_p + 1}$

Back to general automated debugging

$$P(l, a|D) \propto P(D|l, a)P(l, a)$$

- More generally, we would like to infer both the **entire patch** - both the location and the fix action.

- Unlike the space of locations (l), actions are potentially infinite, so it has previously been difficult to come up with different formulae.

One such attempt: Unified Debugging

- As an example, we could analyze the unified debugging approach **SeAPR** from Benton et al., whose core assumption is that:

$$P(\exists t.(t = \text{fail} \land t_{(l,a')} = \text{pass})|(l,a) = \text{fix}) = p_1 \tag{8}$$

$$P(\exists t.(t = \text{fail} \land t_{(l',a')} = \text{pass})|(l,a) = \text{fix}) = p_2 \tag{9}$$

- Patches that make a previously failing test pass after patch application are likely to be indicative of the actual patch location (8) and vice versa (9), thus p1 > p2.

Optimal formula under assumptions

$$\log(P((l, a) = \text{fix}|D)) \propto p^+ - \gamma p^- \qquad (10)$$

where p^+ is the number of high-quality patches at l, while p^- is the number of low-quality patches at l, and $\gamma = \frac{\log((1-p_2)/(1-p_1))}{\log(p_1/p_2)}$. This

- …which is equivalent to the Wong2 formula. Unfortunately, Benton et al. didn't experiment with Wong2:

	Tarantula	Ochiai	Ochiai2	Op2	SBI	Jaccard	Kulczynski	Dstar2
Arja	46.23%	40.05%	38.66%	33.57%	53.88%	39.91%	39.91%	40.60%
Avatar	52.16%	54.80%	53.62%	51.74%	52.16%	55.22%	55.22%	53.55%
Cardumen	-7.32%	-7.32%	-7.32%	-7.32%	-13.41%	-7.32%	-7.32%	-7.32%

(From Benton et al., *Self-Boosted Automated Program Repair*, 2021)

Recap of framework

1. As a means of analyzing automated debugging techniques, we proposed using Bayesian inference.

2. With the framework, we could **derive** an SBFL formula proven to to be maximal from first principles alone.

3. Furthermore, using the framework, we could **suggest** a different formula for unified debugging that was better adapted.

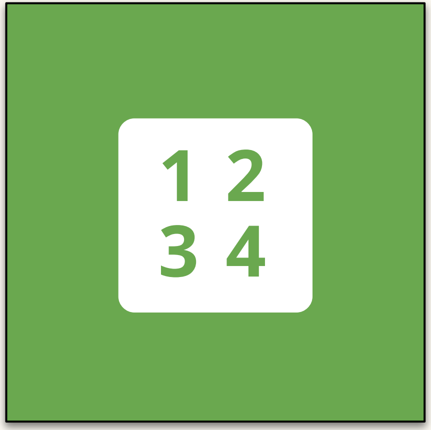
3. Prioritizing Patches

BAPP, a patch prioritization technique

Derived from Framework

Most implementation details are not guessed, but derived

Uses Variable Values



Uses variable values as additional information to prioritize patches

Principle of Value Change

- We observe the value change that would happen when a patch is applied (with debuggers, to efficiently extract values for patches):

$$P(t = \text{failing} | (l, a) = \text{fix} \wedge \text{Ch}(t, (l, a))) = p \tag{13}$$

$$P(t = \text{failing} | (l, a) = \text{fix} \wedge \neg\text{Ch}(t, (l, a))) = 0 \tag{14}$$

- If the correct fix (l, a) would change internal values of t, there is a chance the test would fail (13); if the fix does not induce any chance, a test cannot fail (14).

Final Formula

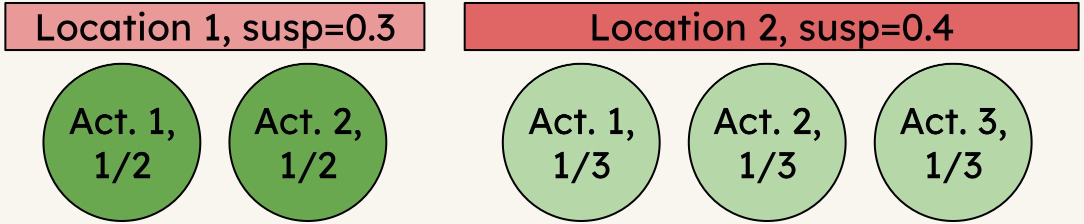
- Skipping a lot of intermediate derivation steps, we get:

$$\log_2 P((l, a) = \text{fix} | D) \propto \begin{cases} -\infty & (c_f < F) \\ \log_2(P(a|l)P(l)) - \alpha c_p & (c_f = F) \end{cases}$$

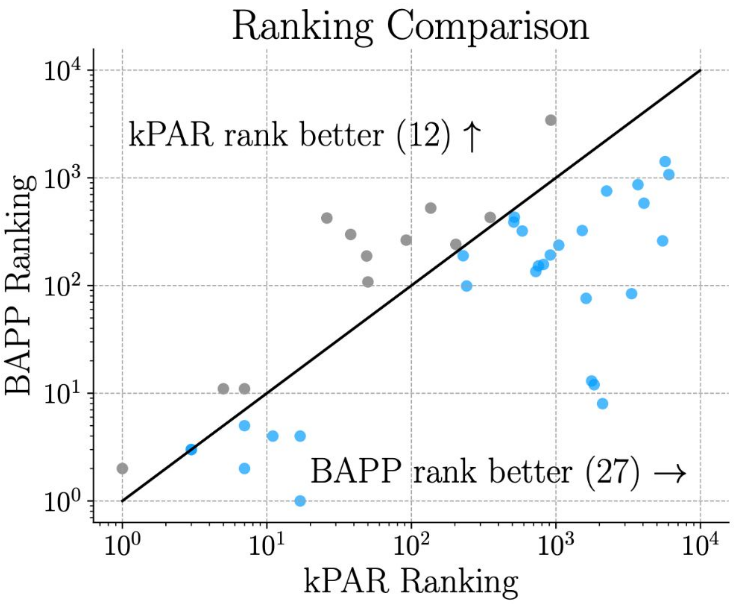
- Where $\alpha = -\log_2(1 - p)$ (note that $\alpha > 0$, as $1 - p < 1$), and intuitively controls how much to weight value change results.
- Fault localization also possible by marginalizing over fix action space.

Other modifications

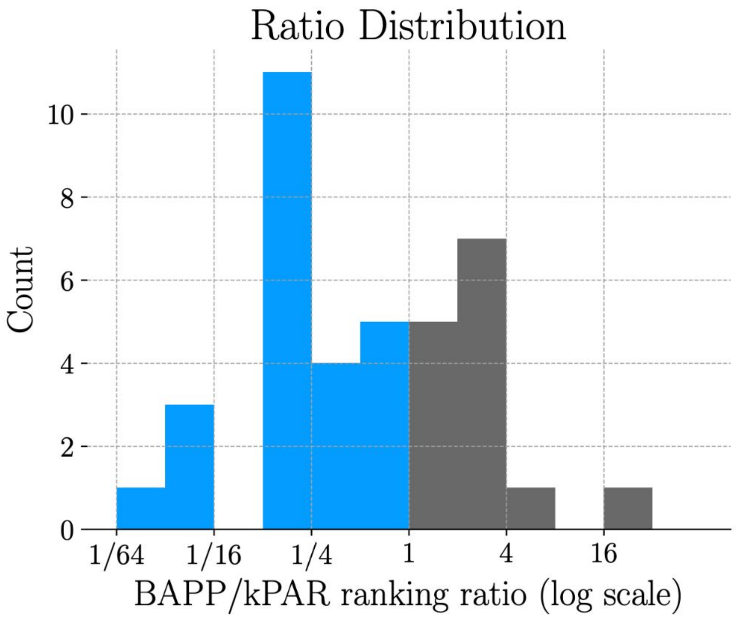
- P(l, a) = P(a|l) **x** P(l): Patch probabilities should be **multiplied** with location probabilities, and sorted accordingly.
- P(l, a) = **P(a|l)** x P(l): but the probability of a patch given a location is often ignored. By modelling this term, we **deprioritize patches from locations with many possible patches**:



RQ1: Patch Prioritization Performance

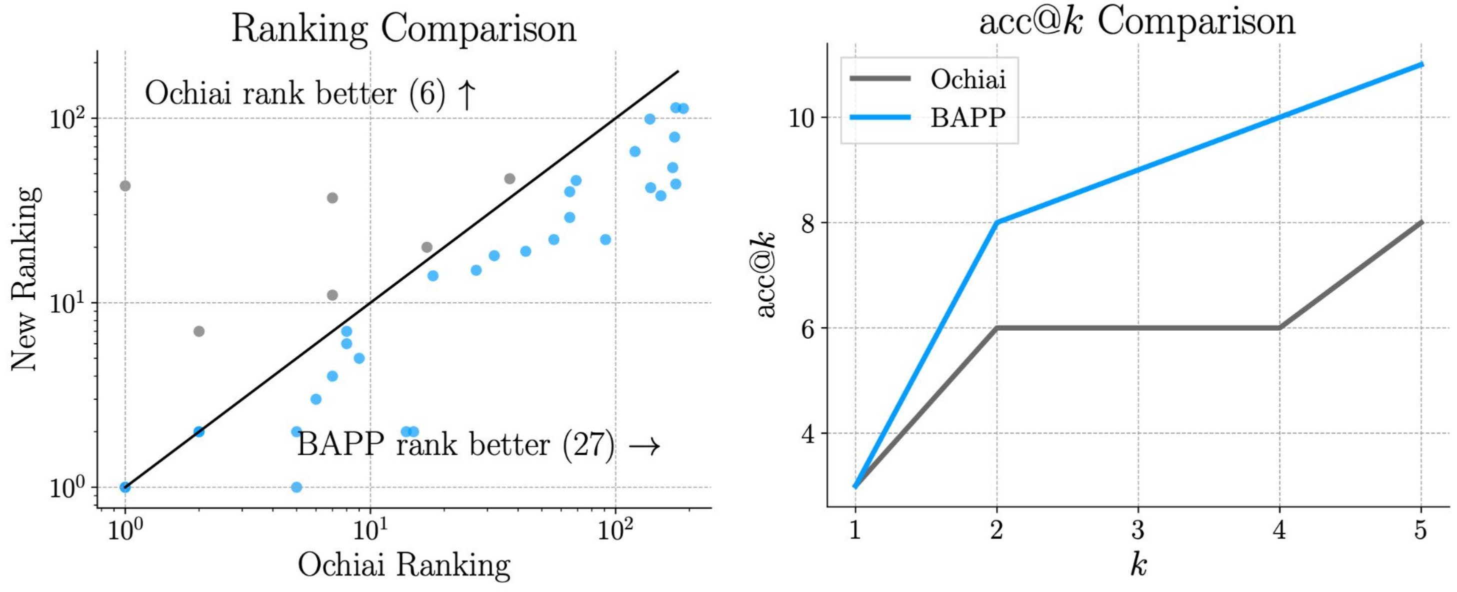


(a) Raw Ranks.



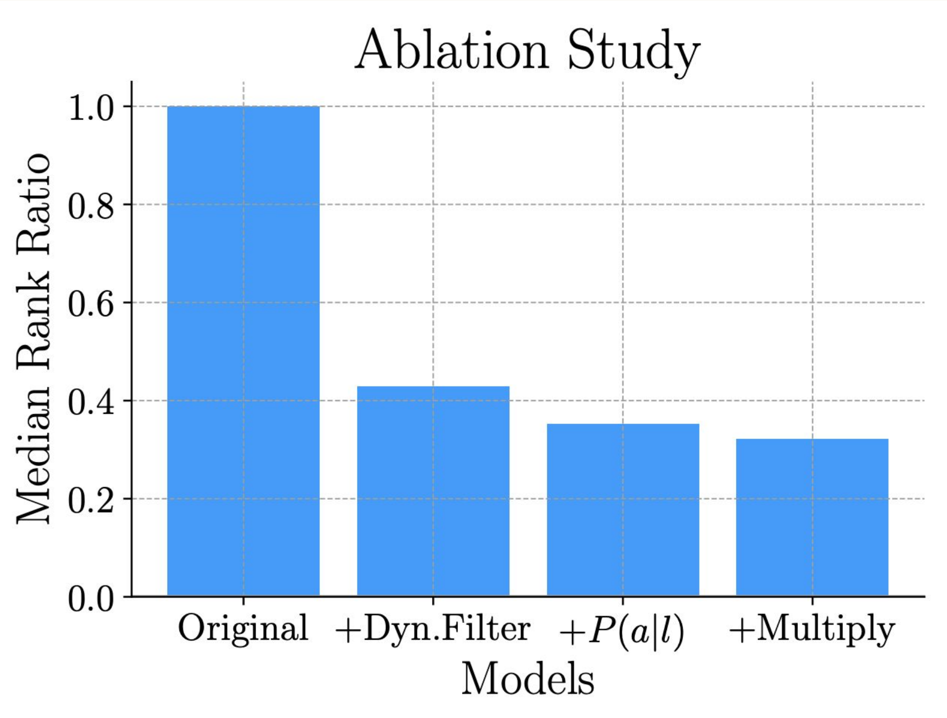
(b) Rank Ratios.

RQ2: Fault Localization Performance



(a) Raw Ranks. (b) acc@k.

RQ3: Ablation Study



4.
Future Work

Framework-suggested future work

1

Patch action identification, then localization: $P(l, a) = P(l|a)P(a)$

2

Modelling observation correlation

3

Efficient inference algorithms for multiple-fault scenarios

4

Incorporation of other information, such as dependency

Conclusion

Read our preprint!

1. Despite the long history of research in **automated debugging**, there was no **theoretic framework** behind the techniques.

2. We propose that **Bayesian inference** can be used to **theoretically analyze** existing techniques.

3. From our theoretic framework, we **derive** a patch prioritization technique, and discuss interesting **future work** stemming from it.

Contact us at sungmin.kang@kaist.ac.kr

Find our preprint with the QR code above, or by searching for “A Bayesian Framework for Automated Debugging”

*

Extra Slides

LLM/ML

- Machine learning, or large language models provide a strong **prior** for debugging results
- However, in my view, **they do not seem to change the need or formulation of the incorporation of new information**
- In this sense, perhaps embeddings are more interesting in the context of my framework, as they might provide means to model result correlation, etc.