

CREST Open Workshop #41

Application of Information Theory to Fault Localisation

Oľ

"How I learnt to stop worrying and love the bomb entropy" by Shin Yoo

This talk is...

- Not a theoretical masterclass on application of Shannon Entropy to software engineering, unfortunately
- Rather a story of a clueless software engineer who learnt to appreciate the power of information theory

The Problem Domain

 Fault Localisation: given observations from test execution (which includes both passing and failing test cases), identify where the faulty statement lies.

1	160	return ret;
1	L61	}
1	L62	
1	L63⊝	<pre>private SolutionSet spreadBetIr</pre>
1	L64	{
1	L65	SolutionSet pop = new Solut
1	L66	DecisionVariables dv;
1	L67	
1	168	//empty selection
1	L69	<pre>dv = new DecisionVariables(</pre>
1	L70	for (int $k = 0$; $k < problem$
1	171	{
01	172	dv.variables_[k] = new
1	L73	}
1	L74	<pre>pop.add(new Solution(proble</pre>
1	L75	
1	176	//full selection
1	L77	<pre>dv = new DecisionVariables(</pre>
1	L78	for (int $k = 0$; $k < problem$
1	L79	{
1	180	$dv.variables_[k] = new$
1	181	}
1	182	pop.add(new Solution(proble
1	183	

Spectra Based Fault Localisation



Spectra-Based Fault Localisation

Structural	Test	Test	Test	Spectrum				Tarantula	Rank
Elements	t_1	t_2	t_3	e_p	e_f	n_p	n_f		
s_1	•			1	0	0	2	0.00	9
s_2	•							0.00	9
s_3	•				0 _	e_f	2	0.00	9
s_4	Tarantula =			_ 1	0 6	$e_f + r$	0.00	9	
s_5					e_p		e_f	0.00	9
s_6				e_1	$p+n_{I}$	p	$e_f +$	n_f 0.33	4
s_7 (faulty)				()	2	1	0	1.00	
s_8	•	•		1	1	0	1	0.33	4
s_9	•	•	•	1	2	0	0	0.50	2
Result	Р	F	F						

How do we evaluate these?

$$\begin{split} Wong2 &= e_{f} - e_{p} \\ \frac{2c_{f}}{e_{f} + n_{f} + e_{p}} \\ Op1 &= \begin{cases} -1 & \text{if } n_{f} > 0 \\ n_{p} & \text{otherwise} \\ e_{f} & \text{otherwise} \\ e_{f} + n_{p} + 2(e_{p} + n_{f}) \\ e_{f} + n_{f} + e_{p} \\ \end{cases} \\ Jaccard &= \frac{e_{f}}{e_{f} + n_{f} + e_{p}} \\ Tarantula &= \frac{\frac{e_{f}}{e_{p} + n_{p} + \frac{e_{f}}{e_{f} + n_{f}}}{\frac{e_{p}}{e_{p} + n_{p} + \frac{e_{f}}{e_{f} + n_{f}}} \\ AMPLE &= |\frac{e_{f}}{e_{f} + n_{f}} - \frac{e_{p}}{e_{p} + n_{p}}| \\ \frac{\frac{e_{f}}{2(e_{f} + n_{p}) + e_{p} + n_{f}}}{2(e_{f} + n_{p}) + e_{p} + n_{f}} \\ \frac{e_{f}}{\frac{e_{f}}{n_{f} + e_{p}}} \\ \frac{e_{f}}{e_{f} + n_{f} + e_{p} + n_{p}} \\ \frac{\frac{e_{f}}{e_{f} + n_{f} + e_{p} + n_{p}}}{\frac{e_{f} + n_{f} + e_{p} + n_{p}}{\frac{e_{f} + n_{f} + e_{p} + n_{p}}}} \\ \frac{e_{f} + n_{f} + e_{p} + n_{p}}{\frac{e_{f} + n_{f} + e_{p} + n_{p}}}} \\ \frac{1}{2}(\frac{e_{f}}{e_{f} + n_{f}} + \frac{e_{f}}{e_{f} + e_{p}}) \\ \frac{1}{2}(\frac{e_{f}}{e_{f} + n_{f}} + \frac{e_{f}}{e_{f} + e_{p}}) \\ Wong3 = e_{f} - h, h = \begin{cases} \frac{e_{f} + n_{p} - n_{f} - e_{p}}{\frac{e_{p}}{e_{f} + n_{f} + e_{p} + n_{p}}} \\ \frac{e_{p} + n_{f} + e_{p} + n_{p}}{\frac{e_{p}}{e_{f} + n_{f} + e_{p} + n_{p}}} \\ \frac{e_{p} + n_{f} + e_{p} + n_{p}}{\frac{e_{p}}{e_{f} + n_{f} + e_{p} + n_{p}}} \\ \frac{e_{p} + n_{f} + e_{p} + n_{p}}{\frac{e_{p}}{e_{f} + n_{f} + e_{p} + n_{p}}} \\ \frac{e_{p} + n_{f} + e_{p} + n_{p}}{\frac{e_{p}}{e_{f} + n_{f} + e_{p} + n_{p}}} \\ \frac{e_{p} + n_{f} + e_{p} + n_{p}}{\frac{e_{p}}{e_{f} + n_{f} + e_{p} + n_{p}}} \\ \frac{e_{p} + n_{f} + e_{p} + n_{p}}{\frac{e_{p}}{e_{f} + n_{f} + e_{p} + n_{p}}} \\ \frac{e_{p} + n_{f} + e_{p} + n_{p}}{\frac{e_{p}}{e_{f} + n_{f} + e_{p} + n_{p}}} \\ \frac{e_{p} + n_{f} + e_{p} + n_{p}}{\frac{e_{p}}{e_{f} + n_{f} + e_{p} + n_{p}}} \\ \frac{e_{p} + n_{f} + e_{p} + n_{p} + n_{p} + e_{p} + n_{p}}{\frac{e_{p}}{e_{f} + n_{f} + e_{p} + n_{p}}} \\ \frac{e_{p} + n_{f} + e_{p} + n_{p}}{\frac{e_{p}}{e_{f} + n_{f} + e_{p} + n_{p}}} \\ \frac{e_{p} + n_{f} + e_{p} + n_{p} + n_{p} + n_{p}}{\frac{e_{p}}{e_{f} + n_{f} + e_{p} + n_{p}}} \\ \frac{e_{p} + n_{f} + n_{f} + e_{p} + n_{p} +$$

Expense Metric

$$E(\tau, p, b) = \frac{\text{Ranking of } b \text{ according to } \tau}{\text{Number of statements in } p} * 100$$

- Assumes that the developer checks the ranking from top to bottom
- The higher the faulty statement is ranked, the earlier the fault is found

Statement Ranking



Does every test execution help you?

- * When a statement is executed by a failing test, we suspect it more; by a passing test, we suspect it less.
- * *Ideally*, we want the failing test to only execute the faulty statement, which is not possible of course.
- *Practically*, we want the subset of test runs that gives us the most distinguishing power, and we want this as early as possible.

What is the information gain of executing one more test?

$$\mathbf{P}_{T_i}(B(s_j)) = \frac{\tau(s_j|T_i)}{\sum_{j=1}^m \tau(s_j|T_i)}$$

Compute the Shannon Entropy of Fault Locality $\mathbf{H}_{T_i}(S) = -\sum_{j=1}^{m} \mathbf{P}_{T_i}(B(s_j)) \cdot \log \mathbf{P}_{T_i}(B(s_j))$

Assuming the failure rate observed $\mathbf{P}_{T_{i+1}}(B(s_j)) = \mathbf{P}_{T_{i+1}}(B(s_j)|F(t_{i+1})) \cdot \alpha +$ so far, compute lookahead P $\mathbf{P}_{T_{i+1}}(B(s_j)|\neg F(t_{i+1})) \cdot (1-\alpha)$

We can predict the information gain of a test case!

steep, with IT ANOP 129



Percentage of Executed Tests

Lessons Learned #1

- Probabilistic view works! Even when there are some wrinkles in your formulations.
- Software artefacts tend to exhibit continuity (e.g. coverage of a test case does not change dramatically between versions, etc). This helps the point 1.

Problem Solved...?

- Various empirical study established partial rankings between formulas at first.
- * Then a theoretical study *proved* the dominance between formulas and their performance in Expense metrics.

But then machines arrived.

Aside: we also automatically evolved formulas using GP, which we then proved cannot be bettered by humans. So technically machines arrived *twice*.

Machine Based Evaluation

- Qi et al. took a backward approach
- Use suspicious score as weights to mutate program states until Genetic Programming can repair the fault.
- * The better the localisation, the quicker the repair will be found.

Using Automated Program Repair for Evaluating the Effectiveness of Fault Localization Techniques

Yuhua Qi, Xiaoguang Mao^{*}, Yan Lei, and Chengsong Wang School of Computer National University of Detense Technology, Changsha, China (yuhua.gi, xgmao, yaniei)@nudLedu.cn jameschen186@gmail.com

ABSTRACT

Many techniques on sutemated hult localization (AFL), have been introduced to assist developers in debugging. Prior studios evaluate the localization technique from the viewpoint of developers: measuring how many benefits that developers can obtain from the localization technique used when debugging. However, these evaluation approaches are not always suitable, because it is difficult to quantify precisely the benefits due to the complex dobugging behaviors of developers. In addition, recent user studies have presented that developers userking with AFL do not correct the defects more efficiently than ones working with only traditional debugging techniques such as breakpoints, even when the effectiveness of AFL is artificially improved.

In this paper we attempt to propose a new research direction of developing AFL techniques from the viewpoint of fully automated debugging including the program repair of automated debugging including the program repair of automation, for which the activity of AFL is necessary. We also introduce the NCP score as the evaluation measurement to assess and compare various techniques from this perspective. Our experiment on 15 popular AFL techniques with 11 subject programs shipping with real-106 field hillinew presents the evidence that these AFL techniques performing well in prior studies do not have better localization effectiveness acoreding to NCP score. We also observe that Jaccard has the better performance over other techniques in our experiment.

Categories and Subject Descriptors

D.2.5 [Software Engineering]: Toxing and Debugging: D.2.3 [Software Engineering]: Software Engineering—Coding Tools and Techniques

General Terms

Experimentation, Measurement

*The corresponding author.

Premission to make digital or hard copies of all or part of this work for prevent or characters out to granted without the provided that copies on not made or distributed for poilt or recompresid advantages and that copies there its notice and for halt estimon the first gaps. Capyrights for components of this work consult by advant that ACM toget the housing Advantage spectra (code) is perturbed. To copy distribute, or regulation, tage the housing Advantage spectra (code) is perturbed for the residence and the line. Required presentations from Promotional Research, and the Required presentation from Promotional Research.

03727/73, July 15-20, 2013, Lagaro, Switzerland Copyright 2015 ACM 978-1-4905-2159-4/1507_515:00 http://dx.doi.org/10.1145/2483760.2483785 Keywords

Fault localization, automated program repair, automated debugging

1. INTRODUCTION

Automated fault localization (AFL) techniques are devel oped to reduce the effort of software debugging, which is well known to be frustrating, and often time-consuming [34] Rather than attempting to pinpoint and suggest a fix for a bug, most existing techniques on AFL give the programmer uggestions about likely fault locations, such as a suspelist (ranking program entities according to their likelihood of containing the fault) produced by statistical approaches, or a small fraction of code (to which failure cause is narrowed down) generated by experimental approaches [26]. Since lots of techniques have been presented in the past decade, one popular research area is to evaluate and compare the effectiveness of various AFL techniques. In addition, when a novel AFL technique is proposed, it is a common practice to evaluate the effectiveness of this technique, and present the advantage by comparing it with other AFL techniques [3]. Much caleting literature [32, 20, 3, 14] evaluates the efctiveness of AFL from the viewpoint of developers, that is, measuring how many henefits that developers can obtain m the AFL technique used when debugging. To quantify the henefits, much existing work evaluates the effectiveness according to the percentage of the program code that needs to be examined before the faults are identified, which is rered to as the EXAM score [32, 3, 33]. Obviously, a lower EXAM score indicates a better localization effectiveness This kind of evaluation measurement, however, is based on the strong assumption of "perfect bug detection" that examining a faulty statement in isolation is enough for a developer to understand and eliminate the bug. Unfortunately, this simplistic view of the debugging process does not hold in practice, because understanding the root cause of a failure for developers typically involves complex activities [21]. Hence, existing evaluation approaches are not always suitable in practice due to this strong assumption.

What is worse, given the maturity of the field of AFL, the fact that most developers are still unvilling to debug faulty pergrams through existing fault localizers [34] results in the natural doubt about the rationality of current evaluation measurement. Artually, to our knowledge most developers are more likely to proceed using traditional debugging such as breakpoints (which can provide data values for interesting statements to assist fault understanding), and then analyze the possible reason through complete human superimer able

Strange Results

- * Theory says Jaccard formula is worse than Op2.
- But machines found it much easier to repair programs when using the localisation from Jaccard.
- * Why?

Abstraction destroys Information

- Expense metric assumes linear consumption of the result (i.e. developer checks statements following the ranking).
- GP consumes raw suspiciousness numbers, which is a much richer source of information.



Same ranking, completely different amount of information.

New Evaluation Metric

- Following the way we predicted information yield, we should be able to describe the true fault locality as a probability distribution.
- Subsequently, measure the cross-entropy between the true distribution and one generated by any technique.

$$\mathcal{L}(s_i) = \begin{cases} 1 & (s_i = s_f) \\ \epsilon & (0 < \epsilon \ll 1, s_i \in S, s_i \neq s_f) \end{cases}$$

$$P_{\tau}(s_i) = \frac{\tau(s_i)}{\sum_{i=1}^{n} \tau(s_i)}, (1 \le i \le n)$$

$$D_{KL}(P_{\mathcal{L}}||P_{\tau}) = \sum_{i} \ln \frac{P_{\mathcal{L}}(s_i)}{P_{\tau}(s_i)} P_{\mathcal{L}}(s_i)$$

Locality Information Loss (LIL) defined with Kullback-Leibler divergence

Worth a thousand words.



Lessons Learned #2

- * Entropy measures are much richer than simply counting something: it gives you a holistic view.
- Cross-entropy is a vastly underused tool in software engineering in general.

Spectra Based Fault Localisation



grep, v3, F_KP_3



Percentage of Executed Tests