BayesFLo: Bayesian Fault Localization for Software Testing

Yi Ji[1,*], Ryan Lekivetz[2], Simon Mak[1] and Joseph Morgan[2]
[1]Department of Statistical Science, Duke University, Durham NC, USA
[2]JMP Statistical Discovery LLC, SAS Institute Inc., Cary NC, USA
yi.ji@duke.edu, Ryan.Lekivetz@jmp.com, sm769@duke.edu, Joseph.Morgan@jmp.com
*corresponding author

2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security Companion (QRS-C) | 979-8-3503-5939-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/QRS-C60940.2023.00019

Abstract—Fault localization is a software testing activity that is critical when software failures occur. We propose a novel Bayesian fault localization method, yielding a principled and probabilistic ranking of suspicious input combinations for identifying the root causes of failures.

Keywords–Fault Localization; Bayesian Modeling; Software Testing.

1. Introduction

Combinatorial testing can greatly reduce testing costs while increasing the likelihood of detecting faults [1]. However, when failures occur, a test engineer must go through the arduous task of trying to identify the root cause(s) of such failures. This is known as the *fault localization* problem. A comprehensive review of this problem and existing solutions is provided in [2]. Relevant methods include [3], who proposed a "SOBER" method to rank predicates using computed scores without prior knowledge. [4] further integrated prior beliefs of test engineers for fault localization by assigning weights to inputs. Our approach, inspired by [4], provides a principled and probabilistic ranking of suspicious input combinations via a novel Bayesian modeling paradigm.

2. Methodology

BayesFLo consists of three important modeling steps: prior specification, likelihood evaluation and posterior ranking.

2.1. Prior Specification

We first define notation. Consider a software system with multiple inputs, denoted by letters $A, B, C, \cdots$. Each input has multiple levels, denoted by numbers $1, 2, 3, \cdots$. The j-th level of input i is denoted as ij, e.g., different levels of input A are written as $A1, A2, \cdots$. Combinations of multiple inputs can then be denoted as $i_1j_1i_2j_2\cdots$, e.g., the two-input combination of $A1$ and $B2$ is $A1B2$. For simplicity, we use $ij^{(k)}$ to denote an arbitrary k-input combination, and ij (rather than $ij^{(1)}$) for an individual input/level. We further use $p_{ij^{(k)}} \in [0,1]$ to denote the prior root cause probability of input combination $ij^{(k)}$, and the random variable $Z_{ij^{(k)}} \in \{0,1\}$ to denote the root cause indicator of $ij^{(k)}$. For $k = 1$, this is again simplified to p_{ij}, Z_{ij}.

We propose a two-step approach for prior specification, by first (i) assigning priors to individual inputs and then (ii) propagating them to multi-input combinations. In step (i), we adopt independent Beta priors on the root cause probability p_{ij} for individual input ij:

$$p_{ij} \sim \text{Beta}(a_{ij}, b_{ij}), \tag{1}$$

where a_{ij}, b_{ij} are the shape parameters of the Beta distribution. These parameters can be set to reflect prior knowledge of faults in the system. For instance, one expects input ij to have relatively higher chance of failure, then the parameters a_{ij} and b_{ij} can be set so that the expectation $\mathbb{E}(p_{ij}) = a_{ij}/(a_{ij} + b_{ij})$ is relatively greater. This provides user flexibility for integrating prior knowledge for probabilistic fault localization.

The root cause indicator Z_{ij} is then modeled as a Bernoulli distribution with probability p_{ij}, where a "success" here corresponds to input ij being the root cause of failure. The distribution of Z_{ij} *conditioned* on p_{ij} can be written as:

$$Z_{ij}|p_{ij} \sim \text{Bernoulli}(p_{ij}). \tag{2}$$

The *marginal* prior root cause probability for input ij (after marginalizing out uncertainty in p_{ij}) becomes:

$$P(Z_{ij} = 1) = \int_0^1 p_{ij} f(p_{ij})\, dp_{ij} = \mathbb{E}(p_{ij}) = \frac{a_{ij}}{a_{ij} + b_{ij}}, \tag{3}$$

where $f(p_{ij})$ is the Beta probability density function for p_{ij}.

Step (ii) of prior specification involves propagating the priors from individual inputs to combinations of multiple inputs. We adopt the following propagation rule:

$$p_{ij^{(k)}} = \prod_{mn \in \text{Pa}_{\text{ind}}(ij^{(k)})} p_{mn}, \tag{4}$$

where $\text{Pa}_{\text{ind}}(ij^{(k)})$ denote the set of k individual inputs mn that are "parent inputs" of $ij^{(k)}$. For example, $\text{Pa}_{\text{ind}}(A1B2C1) = \{A1, B2, C1\}$. With this, the marginal prior root cause probability for the combination $ij^{(k)}$ becomes:

$$\mathbb{P}(Z_{ij^{(k)}} = 1) = \prod_{mn \in \text{Pa}_{\text{ind}}(ij^{(k)})} \frac{a_{mn}}{a_{mn} + b_{mn}}. \tag{5}$$

A key appeal for the propagation rule (4) is that it captures the desired combination principles introduced in [6]. First, since the prior probability for k-input combinations $p_{ij^{(k)}}$ is the product of those for its parents, it follows that $p_{ij^{(k)}} \leq p_{mn}, \forall mn \in \text{Pa}_{\text{ind}}(ij^{(k)})$, which satisfies the combination *hierarchy* principle [6]. Second, note that the prior failure probability of the combination $p_{ij^{(k)}}$ is large only when those for its parents are all large; this thus satisfies the combination *heredity* principle [6].

2693-9371/23/$31.00 ©2023 IEEE
DOI 10.1109/QRS-C60940.2023.00019

2.2. Likelihood Evaluation

After prior specification, we now consider the likelihood of $\{Z_{ij^{(k)}} = 1\}$, the event that the k-input combination $ij^{(k)}$ is indeed the root cause of failure[1]. Conditioning on the observed data (i.e., test cases with outcomes), we can then classify the tested input combinations into 2 disjoint categories:

- TP: Tested and Passed combinations, not root cause,
- TF: Tested and Failed combinations, potential root causes.

We next define $\mathcal{C}_P$ as the set of all input combinations covered in the passed test cases, and $\mathcal{C}_F$ as all combinations covered in the failed test cases. With this, it follows that:

$$\text{TP} = \mathcal{C}_P, \quad \text{TF} = \mathcal{C}_F \backslash (\mathcal{C}_P \cap \mathcal{C}_F). \tag{6}$$

2.3. Posterior Calculation

The last step of BayesFLo involves the computation of posterior root cause probabilities of each input combination. Let $\mathcal{D}$ be the observed data. For TP combinations, at least one test case containing such combinations has passed, thus:

$$P(Z_{ij^{(k)}} = 1|\mathcal{D}) = 0, \quad ij^{(k)} \in TP. \tag{7}$$

For TF combinations, we define two complementary events:

- $E = \{\text{at least one TF combination is root cause}\}$,
- $E^C = \{\text{none of the TF combinations is root cause}\}$.

With this, we can calculate each probability as follows[2]:

$$\begin{aligned} \mathbb{P}(E^C) &= \int \prod_{mn^{(k)} \in TF} \left[1 - \left(\prod_{rs \in \text{Pa}_{\text{ind}}(mn^{(k)})} p_{rs} \right) \right] d\mathcal{P}, \\ \mathbb{P}(E) &= 1 - \mathbb{P}(E^C), \end{aligned} \tag{8}$$

where $d\mathcal{P}$ is the product of Beta densities for parents p_{rs}. Applying the law of total probability, we get:

$$\begin{aligned} \mathbb{P}(Z_{ij^{(k)}} = 1) &= \mathbb{P}(E)\mathbb{P}(Z_{ij^{(k)}} = 1|E) + \mathbb{P}(E^C) \cdot 0 \\ &= \mathbb{P}(E)\mathbb{P}(Z_{ij^{(k)}} = 1|E). \end{aligned} \tag{9}$$

The desired posterior root cause probabilities thus become:

$$\mathbb{P}(Z_{ij^{(k)}} = 1|\mathcal{D}) = \mathbb{P}(Z_{ij^{(k)}} = 1|E) = \frac{\mathbb{P}(Z_{ij^{(k)}} = 1)}{\mathbb{P}(E)}. \tag{10}$$

These computed probabilities provide a probabilistic means for ranking potential root causes for diagnosis, where combinations with greater posterior probabilities are more likely to cause the failure.

3. Illustrative Example

We demonstrate the effectiveness of BayesFLo with a 3-input, 2-level example, using a 4-run strength-2 covering array (denoted $\text{CA}(4, 2, 2^3)$) as test cases. Details of covering arrays for combinatorial testing can be found in [1], [4], [6]. We set $A1C2$ as the (sole) root cause in the system, then assign outcomes (pass/fail) to each test case (see Table 1).

[1]Individual input root causes ($Z_{ij} = 1$) are not considered here since they are assumed to have been fixed at an earlier stage of development.

[2]We assume that there is only one failure. The complexity of the calculations is beyond the scope of this paper if there are multiple failures.

Table 1. Test cases with $\text{CA}(4, 2, 2^3)$ design.

A	B	C	Outcome
1	1	1	Pass
2	2	1	Pass
2	1	2	Pass
1	2	2	Fail

Table 2. Posterior probabilities for potential root causes.

Rank	Input Combination	Posterior Probability
1	A1C2	0.49
2	A1B2	0.24
2	B2C2	0.24
4	A1B2C2	0.04

Suppose that, based on prior knowledge, test engineers believe that inputs A and C are more likely to cause failure than input B. We thus apply different priors (see below), which gives higher expected prior probabilities on inputs A and C:

$$\begin{aligned} &p_{A1}, p_{A2}, p_{C1}, p_{C2} \sim \text{Beta}(2, 8), \\ &p_{B1}, p_{B2} \sim \text{Beta}(2, 18). \end{aligned} \tag{11}$$

Table 2 summarizes the posterior root cause probabilities returned by the proposed BayesFLo method. We see that $A1C2$ (the true root cause combination) has the highest posterior root cause probability amongst 4 potential root causes. This shows that our approach indeed provides a reasonable probabilistic ranking of potential root causes, by leveraging available prior information within an intuitive Bayesian modeling paradigm.

4. Conclusion

We have presented here a novel Bayesian fault localization method, called BayesFLo, which integrates domain knowledge of test engineers and provides a principled probabilistic ranking of potential root causes for software testing. Ongoing work includes the use of the computed posterior probabilities for risk analysis and sequential test case generation.

References

[1] Colbourn, C. J. (2004). Combinatorial aspects of covering arrays. Le Matematiche, 59(1, 2), 125-172.

[2] Wong, W. E., Gao, R., Li, Y., Abreu, R., Wotawa, F., & Li, D. (2023). Software Fault Localization: an Overview of Research, Techniques, and Tools. Handbook of Software Fault Localization: Foundations and Advances, 1-117.

[3] Liu, C., Fei, L., Yan, X., Han, J., & Midkiff, S. P. (2006). Statistical debugging: A hypothesis testing-based approach. IEEE Transactions on software engineering, 32(10), 831-848.

[4] Lekivetz, R., & Morgan, J. (2018, July). Fault localization: analyzing covering arrays given prior information. In 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C) (pp. 116-121). IEEE.

[5] Gelman, A., Carlin, J. B., Stern, H. S., & Rubin, D. B. (1995). Bayesian data analysis. Chapman and Hall/CRC.

[6] Lekivetz, R., & Morgan, J. (2021). On the Testing of Statistical Software. Journal of Statistical Theory and Practice, 15(4), 76.