2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)

Supervised Tie Breaking in Test Case Prioritization

Sepehr Eghbali*, Vinit Kudva*, Gregg Rothermel† and Ladan Tahvildari*
*University of Waterloo, †North Carolina State University
{sepehr.eghbali, ladan.tahvildari}@uwaterloo.ca, {vinit.kudva, gerother}@gmail.com

***Abstract*—Test case prioritization reorders sequences of test cases with the aim of increasing the rate at which faults can be detected. Most existing prioritization techniques employ coverage information gathered on previous test case executions to rank test cases. Existing studies in the literature, however, show that there is a high chance that "ties" occur during the prioritization procedure when using coverage-based techniques; that is, there is a high chance that cases will occur in which two or more candidate test cases have identical code coverage behaviors. To break such ties, most techniques resort to random re-ordering of test cases, which can degrade the rate of fault detection. In this work, we use an ensemble of defect prediction models to guide prioritization techniques towards breaking such ties by re-ordering test cases in terms of the likelihood that they will cover fault-prone units of code.**

***Index Terms*—Test case prioritization, tie breaking, fault prediction**

I. Introduction

Test Case Prioritization (TCP) attempts to find an ordering of test cases that maximizes the rate of fault detection. TCP has been an active field of research in software engineering for more than two decades and myriad techniques have been proposed for performing it (see [1] for an overview). The *Total Technique* (TT) and *Additional Technique* (AT) are two extensively studied greedy coverage-based algorithms [2] which select a test case per iteration. On each iteration, considering test cases not yet prioritized, TT selects a test case with the highest coverage whereas AT selects a test case having maximal coverage of entities (code components) not yet covered by previously prioritized test cases. Nevertheless, both techniques may encounter *ties* during their execution. A tie occurs when more than one test case achieves the highest score on a given iteration. Surprisingly, such ties occur with high probability; studies have shown that more than 80% of the iterations in the AT algorithm encounter ties [3].

In this work, we propose a tie breaker approach that incorporates estimates of the fault-proneness of software units. Our main idea is to guide the TCP algorithm towards testing fault-prone units when a tie occurs. Intuitively, because faults are not equally distributed among software units [4], test cases that cover fault prone units should tend to reveal more faults. Our approach, dubbed *Supervised Tie Breaker* (STB), employs an ensemble of base classifiers to detect fault-prone units and then uses this information to break the tie in a supervised fashion. The empirical results of our experiments indicate that using supervision in tie breaking can substantially boost the rate of fault detection.

II. Supervised Tie Breaker

Our proposed approach involves two main components: *Fault Predictor* and *TCP Tie Breaker*. As the names imply, the first component is responsible for defect prediction, whereas the second component prioritizes test cases based on coverage information and the output of the first component. In our approach the fault predictor is an ensemble learner which aggregates the output of multiple base learners in order to achieve a robust prediction.

For efficiency and simplicity, our ensemble classifier uses a weighted averages technique to combine the output of different classifiers in which the weights of each of the classifiers is calculated using their F scores defined as: $F = 2.\frac{\text{precision}\times\text{recall}}{\text{precision + recall}}$. The F score considers both recall and precision and is in fact the harmonic mean of recall and precision. Using ensemble predictors allows for detection of a larger number of fault trends without the need for expert knowledge. Other studies demonstrate the advantages of using ensemble learning to more accurately detect fault-prone units [5].

The ensemble classifier computes the probability of a unit such as O being faulty by linearly combining the output of each classifier: $P_{ensemble}(O) = \frac{\sum_{j=1}^{q} F_j \times P_j(O)}{\sum_{j=1}^{q} F_j}$ where $P_j(O)$ and $P_{ensemble}(O)$ denote the likelihood of unit O being faulty, calculated by the j-th classifier and the ensemble learner respectively and F_j denotes the F score of the j-th classifier.

The *TCP tie breaker* uses the ensemble classifier predictions to strategically break the tie. In particular, given the coverage information $\mathbf{C}$ and the probabilities $P_{ensemble}(O)$s, if a coverage-based TCP technique faces a tie, the proposed tie breaker computes the scores of candidate test cases involved in the tie using the scoring function $score(t) = \sum_{o\in\mathcal{O}} \mathbf{C}_{to} \times P_{ensemble}(O)$, where t denotes the test case and $\mathbf{C}_{ij}$ indicates the parentage of unit j covered by test case i and the test case with the highest total score wins the tie. Given the scores for each candidate test case involved in the tie, the prioritization algorithm proceeds by selecting the test cases with the maximum score.

An astute observer might point out that instead of calculating scores only when a tie occurs, one can prioritize the entire test suite by using the scoring function in every step. In fact, this idea lies at the core of the prioritization technique proposed recently by Wang et al. [6]. As shown in our empirical study, however, such an approach can produce inferior results presumably because coverage has a higher correlation with the number of faults detected by a test case.

2574-1934/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSE-Companion.2019.00095

TABLE I: Details of the selected projects

Project Name	Language(s)	# Commits	# Classes	# Fixes	# Total Test Cases	# Fault-detecting Test Cases	Git URL
Cassandra	java, python	24076	13330	3222	389	155	git-wip-us.apache.org/repos/asf/cassandra.git
Tomcat	java	21440	6130	6183	533	152	github.com/apache/tomcat.git

III. Empirical Study

We selected Cassandra and Tomcat as objects of study and we analyzed their git repositories to collect data. Details for each of the projects, languages and commits are presented in Table I. The fault predictors as well as the prioritization algorithms used in our research are all applied at the class level granularity merely because the information provided about the commits is at the class level. In this study, we consider each commit containing the word "fix" to correspond to a single fault; therefore, the number of total faults is equal to the number of fix commits. Also, all modified classes reported in a fix commit are considered faulty, and each is considered to contain the same fault. Finally, the test cases reported in the commit are assumed to able to detect the faults reported in the commit. To compute the set of metrics for each software unit, we used CVSAnaly for each project and gathered a total of 94 different metric values on each class. Most of these metrics are part of known metric suites such as Halstead, McCabe and Chidamber & Kemerer.

We train four different classifiers to create the set of base learners, namely Naive Bayes, K Nearest Neighbor, Bernoulli Bayes and Multinomial Bayes (the data is split into training, 5% for validation and the rest for testing). For each classifier, we consider two variants, one that is trained on the original metrics and the other that is trained on metrics transformed by *Principle Component Analysis* (PCA). Therefore, in addition to using different classification models, we also adopt different pre-processing phases (with and without PCA) to obtain diversity in the classifiers.

A. Prioritization Techniques

Although STB is not designed for a specific TCP algorithm, in our study we use it with an additional prioritization technique (AT) that has been shown to be one of the most effective approaches for TCP [7]. In the following experiments, ATTB (AT with Tie-Breaker) denotes the AT algorithm combined with our proposed tie-breaker. We also compare our technique with the recently proposed *Quality-Aware Test Case Prioritization* (QTEP) technique [6]. The idea behind QTEP is to use a linear combination of faults and coverage for every step of prioritization.

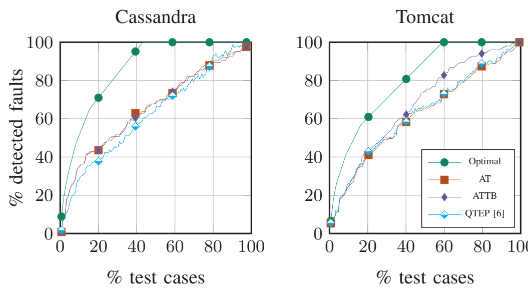

Fig. 1: APFD values of different techniques.

TABLE II: APFD values

Mechanism	Cassandra	Tomcat
AT	61.6%	62.0%
ATTB	65.4%	**67.5%**
QTEP [6]	62.1%	62.5%

B. Results

To measure the effectiveness of different prioritization techniques, we used the Average Percentage Faults Detected (APFD) metric, which is a widely-used metric for evaluating TCP techniques. Table II shows the APFD values obtained for the proposed tie breaking algorithm when used alongside the AT and QTEP. The results show that adopting the proposed tie-breaker can boost the APFD for both projects compared to AT and QTEP. The improvement varies, however, on each project. QTEP performs on par with AT showing only a marginal improvement. Also Fig. 1 plots the rate of fault detection for different techniques; this shows that AT with a tie breaker consistently outperforms original AT and QTEP.

References

[1] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[2] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, 2001.

[3] S. Eghbali and L. Tahvildari, "Test case prioritization using lexicographical ordering," *IEEE Transactions on Software Engineering*, vol. 42, no. 12, pp. 1178–1195, 2016.

[4] T. J. Ostrand and E. J. Weyuker, "The distribution of faults in a large industrial software system," *Software Engineering Notes*, vol. 27, no. 4, pp. 55–64, 2002.

[5] I. H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected features," *Information and Software Technology*, vol. 58, pp. 388–402, 2015.

[6] S. Wang, J. Nam, and L. Tan, "Qtep: quality-aware test case prioritization," in *Proceedings of the Joint Meeting on Foundations of Software Engineering*. ACM, 2017, pp. 523–534.

[7] L. Zhang, D. Hao, L. Zhang, G. Rothermel, and H. Mei, "Bridging the gap between the total and additional test-case prioritization strategies," in *Proceedings of the International Conference on Software Engineering*. IEEE, 2013, pp. 192–201.