Investigating Marginal Contributions of Tests to Coverage Scores

Daniel Gaston
University of Delaware
Newark, DE, USA
dgaston@udel.edu

Patrick Gagliano
University of Delaware
Newark, DE, USA
pgagl@udel.edu

***Abstract*—In this work, we propose a method to investigate the marginal contributions of individual test entities to the overall coverage score attained by a test suite. Our method employs Shapley values, a concept from coalitional game theory, to assign test entities credit according to how much they contributed to a coverage score. Given the well known computational costs associated with the Shapley value, we use an approximation algorithm based on random sampling to estimate Shapley values. Based on an evaluation of our approach using 6 open source Junit test suites, we conclude that Shapley values provide interesting insight into the design and inner workings of test suites.**

***Index Terms*—game theory, software testing**

I. Introduction

When testing software, developers must be informed of the effectiveness of their tests. The most common method for accomplishing this is the use of some coverage metric, such as statement, branch, or mutation coverage [1]. Statement coverage is the number of statements executed, divided by the number of statements in the program, and branch coverage has an analogous definition. Mutation coverage is a measurement of how many artificially seeded faults (mutants) were discovered (killed) by the test suite [2]. Coverage metrics can be used to find deficiencies in the test suite and improve these areas of the suite. For example, if a coverage tool reports that a branch is uncovered, a developer can investigate whether or not a test should be written to cover that branch.

However, these metrics do not tell us about the *contribution* that each test entity (e.g. a test class, test method, all tests in a package, etc) made towards a coverage score. Knowing this will allow us to understand the importance of each test entity in relation to the rest of the suite. In addition to helping us understand past decisions made about a test suite, this information could allow for future decisions to be made. For example, if two similar test classes have significantly different contributions, developers may want to address this.

We propose the use of Shapley values to establish the contribution of each test to the overall coverage score. Shapley values are a concept from cooperative game theory that represent the marginal contribution of each player to the outcome of the game [3]. This allows players to receive a fair reward for their contribution. Previously, Shapley values have been used to determine the marginal contributions of individual features to machine learning models [4, 5], but no work to the best of our knowledge has investigated the use of Shapely values with software test suites.

We present our results as an exploratory step towards the use of Shapley values as an interesting metric to apply to test suites.

II. Approach

In order to calculate the marginal contribution of a test entity, our approach requires a mapping of test entities to achieved coverage objectives (e.g. each test method associated with its covered lines). In this work we will define test entities as test methods and test classes, and coverage objectives as mutants.

A. Shapley Value Calculation

The Shapley value tells us how to fairly distribute the payout of a game among players [3]. Given a game of (N, v) where N is the number of players and v is the value of the game, the marginal contribution of player i can be defined as:

$$\phi_i(v) = \frac{1}{N!} \sum_{\pi \in \Pi} v(p_\pi^i \cup i) - v(p_\pi^i) \tag{1}$$

where π is one permutation of all possible permutations Π and p_π^i is the set of players preceding i in permutation π [6]. Thus, Eq. (1) can be interpreted as finding the difference in value when player i has been added to a game and when player i was not in the game. This summation is done for every player in the game for every possible permutation. The average marginal contribution of a player is found by dividing this sum by $N!$, the number of permutations.

For our problem domain, players are test entities, the payoff function $v(p_\pi^i)$ is the coverage obtained by the tests in p_π^i before test i was added and $v(p_\pi^i \cup i)$ is the coverage obtained after i is added. The total payoff of the game (v) is the overall coverage score obtained by all test entities. The advantage of using the Shapley value to evaluate the contributions is that it will allow us to find a precise solution to dissecting the contribution of each test entity. However, the calculating the exact Shapley value is computationally infeasible for even modest numbers of test entities since the problem is NP-complete [7].

In order to reduce computational costs, we adopt a method of randomly sampling from the permutation space. Sampling

TABLE I: Subjects of our evaluation.

	# of Test		# of Mutants	
Subject	**Classes**	**Methods**	**Killed**	**Total**
Apache Collections	112	2,408	3,447	8,446
Apache Math	363	3,854	24,275	33,282
JavaPoet	19	382	959	1,184
JFreeChart	330	2,099	10,949	33,850
jglm	5	52	744	3,801
jsoup	35	717	3,284	5,009

allows us to calculate the Shapley value more efficiently, but in turn produces approximations and not exact values. Prior work has also formulated and used sampling methods in experiments [4, 5, 7, 8, 9]. The formulation for calculating the estimated Shapley value based on sampled permutations is similar to Eq. (1):

$$\phi_i(v) = \frac{1}{S}\sum_{s=1}^{S} v(p_s^i \cup i) - v(p_s^i) \tag{2}$$

where S is the total number of samples taken from the original permutation space and p_s^i is the set of players preceding i in sample s. In practice, we implement this approach by choosing the largest S that is feasible given our computational resources and select a sample s by randomly shuffling the set of players.

III. Evaluation

In order to evaluate our approach, we ask the following research questions:

> *RQ1—Low Granularity Analysis:* Within each test suite, what is the distribution of Shapley values of the test classes?
> *RQ2—High Granularity Analysis:* Within each test class, what is the distribution of the Shapley values of each test method?
> *RQ3—Approximation Error:* How do approximated Shapley values differ from the true values?

A. Subject Selection

We use 6 Java applications as the subjects of our evaluation as shown in Tab. I. The first column is the name of the application and the following two columns show the number of test classes and test methods in the test suite of the application. For example, in jsoup there are 36 test classes and 719 test methods.= The final two columns show the number of mutants killed and the total number of mutants generated. The mutation score of each subject can be calculated by dividing the number of killed mutants by the total number of mutants. For example, in jsoup $3{,}284$ mutants were killed out of a total of $5{,}009$ mutants generated, leading to a mutation score of $65.56\,\%$.

This selection of subjects was made to satisfy a number of criteria. First, they are all publicly available, open source projects, which facilitates replication and extension of this work. Second, the applications come from different subject domains: Apache Collections and Math are general purpose libraries, JavaPoet is a library for generating java source files, JFreeChart is a plotting library, jglm is a library for graphics math, and jsoup is an HTML parsing library. Third, the applications have test suites that have different structures and sizes, which increases the chances that our findings will generalize to other projects. Finally, several of these projects are commonly used in the software testing literature (e.g., [10]), which makes them logical subjects to include.

TABLE II: Standard deviation of Shapley values for all test classes within each subject.

Subject	Standard Deviation
Apache Collections	0.528
Apache Math	0.332
JavaPoet	5.152
JFreeChart	0.128
jglm	2.867
jsoup	2.452

B. Data Collection

We form a dataset of coverage information by using the Pitest mutation coverage tool [11]. Pitest was used due to its easy integration with commonly used build systems, as well as its ability to output a full kill matrix. A full kill matrix is generated when every test is run against every mutant; by default a mutation testing tool stops after any test kills a mutant in order to reduce costs. A full kill matrix is necessary for our approach since we need to know which mutants were killed by which tests. Each kill matrix was parsed and the estimated Shapley value was calculated as described in the previous section. Due to computational constraints, the number of samples was limited to $100{,}000$ for the low granularity analysis and $4{,}000$ for the high granularity analysis. For the remainder of the paper, all Shapley values presented can be interpreted as the number of percentage points that a particular test entity contributed to a coverage score.

C. RQ1—Low Granularity Analysis

The purpose of this research question is to determine how the contributions to the overall coverage score are distributed between test classes. Answering this question will give us a high level insight into how the developers organized the test suite. For example, if we find that all test classes have roughly the same Shapley value, this would indicate that developers consider all test classes to be of equal importance. To answer this question, we calculated the standard deviations for the Shapley values of the test classes within each test suite. In this case, the standard deviation tells us the extent to which the test suites deviate from the assumption that each test class contributes equally to the coverage score.

The results of this calculation are shown in Tab. II. The first column shows the subject, while the second column shows the standard deviation for that subject. For example, JavaPoet has a standard deviation of 5.152 while Apache Math has a standard deviation of 0.332

We can make several interesting observations based on these results. Interpreting the table, we can say that the larger the standard deviation, the more variation there is between the contributions of the test classes. Conversely, small standard deviations mean that most of the contributions of test classes are clustered around the average. We note that the subjects with fewer test classes and methods tend to have higher standard deviations. For example, JavaPoet, jglm, and jsoup are the smallest subjects, but have the highest standard deviations. This makes sense intuitively; the more tests there are, the more chances there are for a mutant to be killed by multiple tests, meaning that the credit for those kills is shared. However, there is still variation between the larger subjects, which could be an interesting avenue for future investigations.

D. RQ2—High Granularity Analysis

The purpose of this research question is similar to that of the first research question, but within the context of individual test classes as opposed to the entire test suite. In other words, we want to find out the contribution of each individual test method to the coverage score obtained by its containing test class. This will provide a more detailed view of the test suite.

To answer this question, we calculated the approximate Shapley value of each test method relative to its test class. For example, if a test class with 2 test methods killed 10 mutants, the Shapley value for each test method would be $50\,\%$ if each test method contributed evenly. If the contributions of the tests were uneven, the values would be different, but would still sum to $100\,\%$, meaning that all of the credit for the mutants killed by a test class is accounted for.

Since test classes kill different numbers of mutants and have different numbers of test methods, we need a way to normalize the Shapley values so that a cross-class comparison can be made. To do this, we calculated the standard deviation of the Shapley values for the test methods within each test class. The number obtained from this calculation can be interpreted as answering the question "How close are the test methods within a test class to having equal contributions to the mutants killed by that test class?" Thus, a standard deviation of 0 indicates that each test method contributed exactly the same amount, whereas a higher standard deviation indicates that one or more test methods contributed more than others. Test classes with only one method were excluded from this analysis, since the contribution made by that method is obvious.

Fig. 1 shows a box plot of the standard deviation values obtained for each test class, per subject. The x axis shows each subject, while the y axis shows the standard deviation. The solid boxes show the interquartile range, which means that a bigger box equates to a wider spread in values. The outliers are marked as triangles above the plots. Additionally, a jitter plot is overlayed on the box plot in order to remind the reader of the difference in size of each subject. Each point represents the value for a single test class. For example, the subject jglm has only 5 points, whereas Apache Math, on the far right, has nearly $4{,}000$

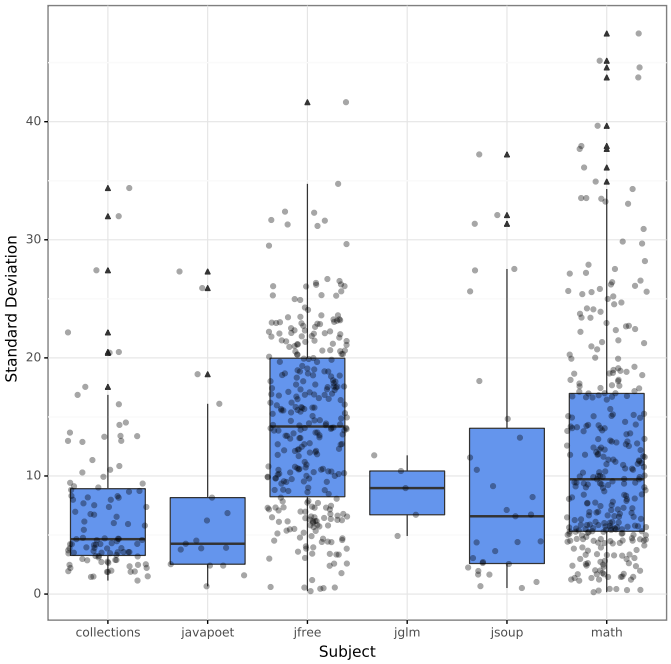

Fig. 1. Standard deviations of Shapley values within test classes.

We also wanted to find out if the standard deviation of test method contributions varies according to the number of test methods in a test class. The results of this calculation are shown in Fig. 2. Each of the six plots shows the result for its respective subject. The x axis of each plot shows the number of test methods in a class, and the y axis shows the standard deviation of the test method contributions. Each point represents the value of a single class. Additionally, there is a trend line drawn over the points to provide a simple visualization. The plot labeled jglm does not have a trend line due to the low number of data points for that subject.

From this data, we can draw the following conclusions: first, there are very few test classes where the credit for mutants kills among the test methods is evenly distributed. In fact, the values of the interquartile ranges in Fig. 1 indicate that most test classes have a handful of "leaders" that kill a disproportionate number of mutants. There are also outliers in every subject but jglm. These test classes are interesting because they contain at least one test method that barely contributes to the mutation score at all. Such methods could serve as a target for developers to improve. Second, Fig. 2 shows that all test classes with a more skewed Shapley value distribution have relatively few test methods. This indicates that the outliers seen in Fig. 1 all contain few methods, and that one or more of those methods contributes most to the mutation score. It is interesting to note that the trend line drawn atop each plot is roughly the same shape. Another interesting point is that test classes with a high number of methods have a more even distribution. This makes sense intuitively, as noted elsewhere, and indicates that there is a lot of overlap in the mutants killed by the test methods within those test classes.

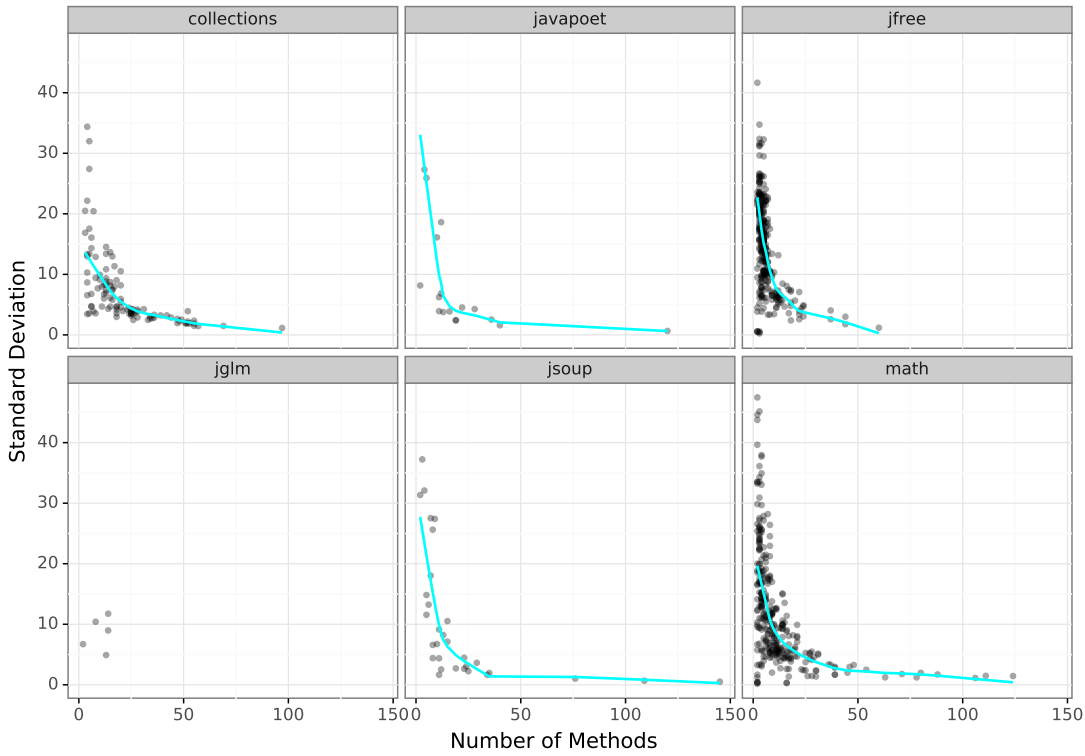

Fig. 2. The standard deviation of the Shapley value within each test class by the number of methods in the class, per subject

E. RQ3—Approximation Error

Since calculating the exact Shapley value for groups of more than a handful of test entities is computationally intractable, we performed random sampling from the permutation space of the test entities, as outlined in our approach. This leads us to ask how much error was introduced because of our approximation.

For the first research question, the only subject for which the exact Shapley value can be reasonably computed is jglm with 5 test classes. We compared this value to the approximation achieved by taking $10{,}000$ samples and $100{,}000$ samples. Since there are only 120 possible permutations, this comparison shows the consequences of oversampling rather than undersampling. The standard deviation of the approximate Shapley values from the exact values calculated with $10{,}000$ samples is 0.013 and 0.003 when calculated with $100{,}000$

We performed a similar analysis to find the error in our high granularity analysis. Since that analysis involved calculating Shapley values *within* each test class, we were able to generate more data, since there are many test classes with few methods. First, we filtered out all test classes with 8 or more test methods, and then calculated the exact Shapley value for each of the methods in the remaining test classes, as described in RQ2. Then we calculated the standard deviation of the estimated Shapley values (based on $4{,}000$ samples) from the exact value. We repeated this calculation 10 times and took the average in order to smooth variations introduced by the random sampling. We found that the average standard deviation ranged from 0.173 to 0.343

Based on these findings, we tentatively conclude that our sampling method produces Shapley values that are reasonably close to the actual values, and that the margin of error likely decreases as more samples are taken. We also suspect that precise Shapley values are not crucial to learn insights from our dataset. For example, the difference between reporting that one test method contributes $30\,\%$ to the mutation score of a class instead of an actual value of $32\,\%$ (an error much larger than what we observed) is immaterial if the remaining test classes all contribute under $10\,\%$. In other words, the conclusion that the contributions of this test method far outweigh the other test methods would not change even for relatively large errors. We leave a more rigorous exploration of the implications of our sampling method to future work.

F. Threats to Validity

Although we tried to mitigate them, there are a few threats to validity for our study. First, our results may not be representative of all Java software projects. To address this, we selected subjects from a variety of subject domains that each have different architectures and test suite design choices. Second, our approximation method could have caused inaccurate results. Although we addressed this concern in our third research question, without a formal analysis these results could still be doubted.

IV. Related Work

A. Shapley Values in Explainable Machine Learning

Similar to our goal of identifying the contributions of test entities to coverage scores at various granularities, much prior work has focused on using Shapley values to determine the contribution of individual features to the prediction of a model [4, 5]. Interpreting predictions from a machine learning model can be quite difficult, and there is often tension between accuracy and interpretability, which makes Shapley values very useful for this domain. Shapley values have also been used to find the most important data in machine learning models [9], as well as the importance of user created tags in social media tagging mechanisms [12].

B. Software Testing

Coverage metrics are widely used for evaluating test suite effectiveness. Among the most common are statement, branch, and mutation testing [1]. The goal of these metrics is to assess the fault finding capabilities of either individual test entities or the entire suite. The goal of our present work is to enhance the information conveyed by coverage information to give developers a more fine grained understanding of their tests.

This work shares similarities with prior work in the software testing field. In particular, test suite reduction is a collection of techniques to find a minimal subset of tests that will meet a particular coverage goal [13]. While these techniques must weigh the contribution of each test towards the coverage goal, this is an internal calculation and is not made available to the developer. This field is also able to use concepts from economics such as finding a Pareto optimal solution [14]. Our goal, on the other hand, is to learn about how test entities contribute to the *existing* coverage score at various granularities.

V. Discussion

In this paper we presented our approach to using Shapley values to estimate the contributions of test entities to coverage scores. Through an empirical investigation, we observed and reported several interesting patterns when our approach was applied to real world software test suites. Notably, we found the presence of many outliers in each test suite, both in terms of the contribution of a test class to the overall coverage score as well as in terms of the contribution of a test method to the coverage score of its class. These outliers could potentially be issues that require developer attention.

A significant drawback of this approach is that it can come with significant overhead. Mutation testing is notoriously expensive, and even more so when the full kill matrix is generated. The Shapley value also requires a large amount of computation, even when sampling is used. Future work could investigate the use of less expensive coverage metrics, as well as trade-offs between execution time and accuracy when calculating the Shapley value.

References

[1] S. Brown, J. Timoney, T. Lysaght, and D. Ye, "Software testing," 2011.

[2] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. Le Traon, and M. Harman, "Mutation testing advances: an analysis and survey," in *Advances in Computers*. Elsevier, 2019, vol. 112, pp. 275–378.

[3] L. S. Shapley, "A value for n-person games," *Contributions to the Theory of Games*, vol. 2, no. 28, pp. 307–317, 1953.

[4] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in neural information processing systems*, 2017, pp. 4765–4774.

[5] M. Sundararajan and A. Najmi, "The many shapley values for model explanation," in *International Conference on Machine Learning*. PMLR, 2020, pp. 9269–9278.

[6] E. Winter *et al.*, "The shapley value," *Handbook of game theory with economic applications*, vol. 3, no. 2, pp. 2025–2054, 2002.

[7] J. Castro, D. Gómez, and J. Tejada, "Polynomial calculation of the shapley value based on sampling," *Computers & Operations Research*, vol. 36, no. 5, pp. 1726–1730, 2009.

[8] S. Maleki, L. Tran-Thanh, G. Hines, T. Rahwan, and A. Rogers, "Bounding the estimation error of sampling-based shapley value approximation," *arXiv preprint arXiv:1306.4265*, 2013.

[9] A. Ghorbani and J. Zou, "Data shapley: Equitable valuation of data for machine learning," *arXiv preprint arXiv:1904.02868*, 2019.

[10] R. Just, D. Jalali, and M. D. Ernst, "Defects4j: A database of existing faults to enable controlled testing studies for java programs," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, 2014, pp. 437–440.

[11] H. Coles, T. Laurent, C. Henard, M. Papadakis, and A. Ventresque, "Pit: a practical mutation testing tool for java," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, 2016, pp. 449–452.

[12] G. G. M. Legesse and D. Teferi, "Selecting feature-words in tag sense disambiguation based on their shapley value," *IEEE*, vol. 1, no. 1, pp. 236–240, 2016.

[13] S. U. R. Khan, S. P. Lee, R. W. Ahmad, A. Akhunzada, and V. Chang, "A survey on test suite reduction frameworks and tools," *International Journal of Information Management*, vol. 36, no. 6, pp. 963–975, 2016.

[14] M. G. Epitropakis, S. Yoo, M. Harman, and E. K. Burke, "Empirical evaluation of pareto efficient multi-objective regression test case prioritisation," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, 2015, pp. 234–245.