Multilevel Coarse-to-Fine-Grained Prioritization for GUI and Web Applications

Dmitry Nurmuradov
Computer Science and Engineering
University of North Texas
Denton, TX, 76203
United States
dn0086@unt.edu

Renée Bryce
Computer Science and Engineering
University of North Texas
Denton, TX, 76203
United States
renee.bryce@unt.edu

Hyunsook Do
Computer Science and Engineering
University of North Texas
Denton, TX, 76203
United States
hyunsook.do@unt.edu

ABSTRACT

This work demonstrates that the use of one criterion for test suite prioritization may lead to high variability of fault detection rates due to random tie-breaking. The paper provides motivational examples of how a single fine-grained or coarse criterion may lead to poor code coverage or fault finding efficiency. We use a multilevel coarse-to-fine-grained two-way prioritization method to address the issues and evaluate the technique in an empirical study by comparing fault finding effectiveness and its variability to single-criterion methods. The results indicate that the proposed method decreases tie-breaking instability of the fault detection rate and often increases the overall performance of test suite prioritization methods.

CCS Concepts

•Software and its engineering → Software testing and debugging;

Keywords

Software Testing; Regression Testing; Test Suite Prioritization; Multiple Criteria Test Suite Prioritization

1. INTRODUCTION

Failures caused by software bugs often have a wide range of implications. Software bugs have led to recalls of automobiles, misrepresentations of votes, cancellations of stock trades, and other problems [1]. About one million software developers in the United States spend their day fixing previously discovered bugs [2]. The use of complex software increases as the computing power of the modern technology increases. More complex software requires more extensive and time-consuming testing. Many techniques exist to help developers reduce software maintenance efforts and improve the testing process [3, 4, 5]. Test suite prioritization is one such technique [6, 7, 8, 9]. The goal of test suite prioritization is to reorder test cases in the way that the given criteria is covered as much as possible and as early as possible. Another testing approach is the concept of user session-based test suites that was introduced by Elbaum et al. [10] to complement traditional white-box methods for web applications. Bryce et al. [8] develop a uniform model for event-driven software and conduct empirical studies that show that the combinatorial two-way test suite prioritization applied to user session-based test suites generally performs better than other methods.

The two-way test suite prioritization, however, is a single-criterion technique, and the use of single-criterion methods may lead to a significant number of tie-breaking cases when multiple tests have the same scores. As a result, the fault detection rates or code coverage produced by single runs of prioritization techniques may not be indicative of the average values that are produced by running the techniques many times. Arcuri and Briand [11] describe a similar problem with randomized algorithms that are often used in the software engineering area. Considering that randomized algorithms are heavily influenced by randomness, a proper statistical analysis is required to obtain reliable results. The authors, however, observe that researchers frequently perform empirical studies with no or weak statistical evidence.

In this work, we demonstrate that test suite prioritization methods often have a component of randomness, which may influence the outcome of experiments. The issue is not limited to test suite prioritization, as tie-breaking frequently occurs in systematic approaches of test case generation, test suite reduction, test case selection, and other software testing techniques. Furthermore, single-criterion prioritization approaches may have poor performance even when random tie-breaking is not an issue. We discuss these issues in greater detail in Section 3.

Several researchers propose prioritization techniques that use two or more criteria. Sampath et al. [12] describe these techniques as hybrid prioritization methods. They identify 44 papers that employ hybrid techniques and classify the techniques into three main categories: *Merge*, *Rank*, and *Choice*.

In this work, we propose the use of a hybrid *Rank* method that is applied to user session-based test suites to study the impact of random tie-breaking compared to systematic tie-breaking. The technique addresses the limitations of single-criterion methods by applying a combinatorial pairwise ap-

A-TEST'16, November 18, 2016, Seattle, WA, USA
ACM. 978-1-4503-4401-2/16/11...$15.00
http://dx.doi.org/10.1145/2994291.2994292

proach using multiple criteria, starting from coarse criteria and increasing granularity at each subsequent level. The main contributions of this paper are as follows:

- We investigate the influence of tie-breaking on a fault detection rate for single-criterion prioritization methods.
- We propose a new prioritization technique that utilizes multiple criteria, from coarse to fine-grained, which addresses limitations of single-criterion techniques.
- We conduct an empirical study using the existing user session-based test suites and show that the proposed method decreases variability of results and often provides a better fault detection rate.

2. RELATED WORK

To date, many test case prioritization techniques have been proposed, but here, we limit our discussion to the techniques that are most closely related to our work.

Fazlalizadeh et al. [13] utilize historical fault detection effectiveness, test case execution history, and historical ranking to prioritize test cases. Their proposed approach is a *Merge* technique as they sum differently weighted scores of multiple criteria. The results show that their method outperforms random ordering. On the contrary, our study compares the multilevel prioritization method to several combinatorial techniques, including two-way inter-window prioritization, in addition to random ordering.

Carlson et al. [14] conduct an industrial study using Microsoft Dynamics as an object application. The authors employ an agglomerative hierarchical clustering approach using criteria such as code coverage, code complexity, fault history information, and a combination of code complexity and fault history. The last criterion shows a study that utilizes a hybrid prioritization technique. The authors omit details regarding how they combined the two criteria. While the results show that the proposed clustering method improves the fault detection rate, the hybrid criteria performed in line with methods that use a single criterion. In our study, we use different types of test suites and criteria and show how the random tie-breaking process affects the results of single-criterion methods.

He and Bai [15] utilize a state-distance-based criterion for test suite prioritization in conjunction with additional event coverage.This is a *Rank* technique that uses additional event coverage as the primary criterion and GUI state similarity as the second. While the authors mention the tie-breaking issue, they do not investigate its influence on the fault detection rate. The authors do not state the frequency of ties or how they address tie-breaking in cases where the secondary criterion produces the equal scores. In our work, we use different criteria for hybrid multilevel prioritization, investigate how random tie-breaking affects the results, and provide statistical evidence that our approach is better than single-criterion prioritization methods.

Sampath et al. [12] generalize hybrid methods into three categories: *Merge*, *Rank*, and *Choice*. They formally define each of the categories for hybrid criteria and classify previous work into those categories. The authors identify 44 papers that employ hybrid prioritization methods. Many of the examined approaches do not analyze the impact of tie-breaking in detail. The authors also propose several hybrid criteria using their formal definitions and evaluate them in the empirical study. In the study, the authors randomly tie-break the test cases when a given criterion is no longer applicable. The study shows that hybrid criteria methods often outperform single-criterion prioritization techniques. Their *Rank* method is comparable to the two-way combinatorial prioritization. For the hybrid *Rank* approach, the authors utilize four single criteria in the given order: two-way combinatorial using parameter/values, the number of parameter/values, the number of windows, and the number of unique windows. Their *Rank* approach uses a fine-grained criterion at first and more coarse criteria for subsequent levels. Their approach does not provide a detailed examination of tie-breaking. In our study, we use a different set of criteria, which follows coarse-to-fine-grained order, repeat experiments 1,000 times, and investigate the influence of random tie-breaking on the fault detection rate.

3. MOTIVATION

3.1 Tie-Breaking Problem

As mentioned in Section 1, Arcuri and Briand [11] investigated 54 publications from top conferences and journals that use randomized algorithms. Only 27 publications contained results from 10 or more runs of the randomized algorithms. The authors state that even some influential papers report results of only one run of a randomized algorithm. As a consequence, generalizability of the reported results in these papers is under a threat.

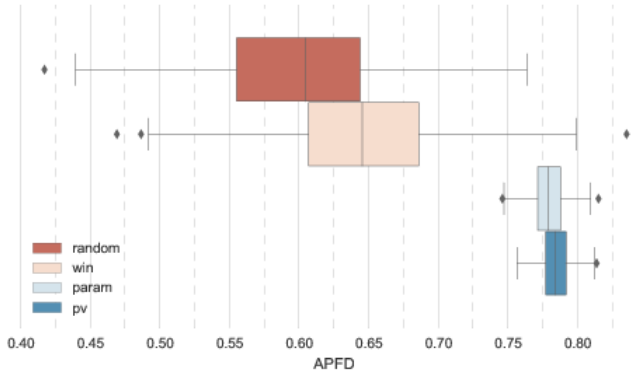

Figure 1: Tie-breaking problem using different combinatorial criteria for two-way prioritization using the TERP Spreadsheet test suite.

In this work, we demonstrate that even systematic methods often contain a component of randomness due to tie-breaking. The tie-breaking issue arises when an algorithm is required to pick one of the candidates that are considered to be equal by a given criterion. For instance, the two-way prioritization technique may have two or more situations when tie-breaking may occur: during the execution, when there are multiple candidates with the same score to be a next test case in an order, and after the execution, when there are still test cases with the same scores. In this study, we explore the latter instance. To provide a motivational example for the issue, one of the object programs was used: TERP Spreadsheet[1]. The detailed description of object programs

[1]https://www.cs.umd.edu/users/atif/TerpOffice/

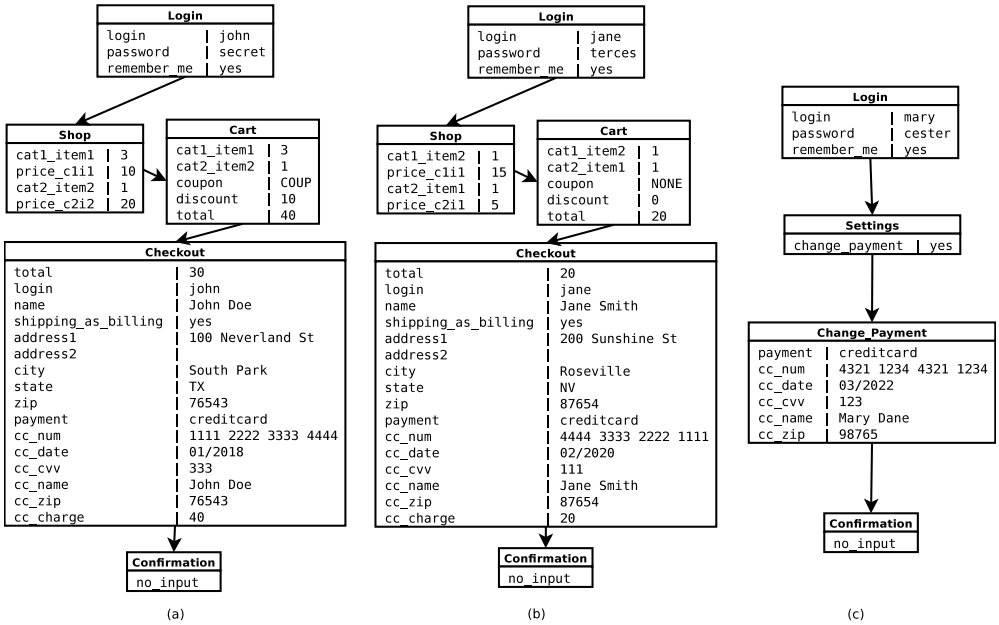

Figure 2: Three user sessions: (a) a user does online shopping; (b) the second user does online shopping, similar to (a); (c) a user changes their payment method.

is available in Section 5.1. TERP Spreadsheet was chosen since its test suite has the lowest fault density among subject applications and has more than 250 test cases.

Figure 1 displays ranges of the average percentage of faults detected (APFD) metric [16] for TERP Spreadsheet using three combinatorial criteria: window (win), parameter (par), and parameter/value (pv) pairs, along with random ordering (random). The APFD metric measures the rate of fault detection in a prioritized test suite and its values range from 0% to 100% with higher numbers indicating faster fault detection rates. Figure 1 demonstrates that the use of window pairs as a prioritization criterion leads to a significant variation in values, ranging from 46.93% to 83.54%, with the standard deviation of 5.7%. The use of parameter pairs or parameter/value pairs still produces the range of more than 2% variance in APFD between minimum and maximum values.

The use of deterministic tie-breaking mechanisms may have an undesirable impact if a non-relevant criterion is chosen. For instance, if a tester decides to break ties using test case identifiers, results may still be in the same wide range of possible values. On the other hand, the use of relevant criteria for breaking ties indicates that a tester employs a hybrid *Rank* prioritization method.

3.2 Fine-Grained and Coarse Criteria

The use of fine-grained criteria may also lead to undesirable results independent of the tie-breaking problem. Consider three test cases demonstrated in Figure 2. Each block in the figure represents a window. Window names are shown at the top of the blocks. In each block, there are parameters and values, which are used to pass information from one block to another. For instance, *Login, Shop, Cart, Checkout*, and *Confirmation* are window names for test case **a**. In window *Login*, there are parameters *login, password*, *remember_me* and values *john, secret*, *yes* correspondingly. Consider applying two-way interaction prioritization [8] using the fine-grained criterion such as pairs of parameter-values to the given example. In test case **a**, there are 3*4 pairwise interactions between parameter-values in *Login* and *Shop* windows, 3*5 between parameters in *Login* and *Cart* windows, etc. The total number of pairwise inter-window interactions between parameter-values in test case **a** is 239. Similarly, there are 239 inter-window interactions in test case **b**. There are 27 pairwise inter-window interactions in test case **c**. When the two-way inter-window prioritization algorithm starts choosing test cases, test cases **a** and **b** have the same scores and test case **c** has a lower score. As a result, the prioritized order is either [**a,b,c**] or [**b,a,c**], depending on tie-breaking. It is likely that the first two test cases will execute similar segments of the code and find the same faults. Test case **c** has a different execution path and may find different faults. To address the issue, testers may consider using more coarse criteria such as two-way interactions between windows. Given that two-way interaction prioritization relies on uncovered criteria to find the next test case for a prioritized suite, the resulting order is [**a,c,b**] or [**b,c,a**] and addresses the issue with fine-grained criteria.

On the other hand, the use of coarse criteria in a single-criterion method may result in a large number of test cases with the same scores. Consider the following example:

- A test suite contains n test cases
- A tester uses the number of windows in a test case as a criterion for prioritization
- The number of windows varies from i to j, where $j > i$ and $d = j - i$
- n is significantly larger than d

There are at most d different scores for prioritization for the given example. Since n is significantly larger than d, a prioritized order effectively will be a partitioned test suite

```
1  function multiLevelPrioritization
       Data:
           unorderedSet
       Result:
           orderedSet
2      pairs ← generateWindowPairs(unorderedSet)
3      orderedSet ← prioritizeTestCases(pairs, unorderedSet)
4      testCaseList ←
       extractTestCasesWithTheSameScore(orderedSet)
5      if testCaseList ≠ ∅ then
6          pairs ← generateParameterPairs(testCaseList)
7          testCaseList ← prioritizeTestCases(pairs,
           testCaseList)
8          orderedSet ← mergeScores(orderedSet, testCaseList)
9          testCaseList ←
           extractTestCasesWithTheSameScore(orderedSet)
10         if testCaseList ≠ ∅ then
11             pairs ←
               generateParameterValuePairs(testCaseList)
12             testCaseList ← prioritizeTestCases(pairs,
               testCaseList)
13             orderedSet ← mergeScores(orderedSet,
               testCaseList)
14         end
15     end
```

Algorithm 1: Description of the proposed algorithm.

with d partitions and randomly ordered test cases inside each of the partitions.

4. MULTILEVEL COARSE-TO-FINE-GRAINED TEST SUITE PRIORITIZATION

To address the issues described in Section 3, we propose the multilevel prioritization algorithm for user session-based test suites. The method is characterized as a *Rank* hybrid technique and pseudo-code is shown in Algorithm 1.

The algorithm is a modification of the two-way inter-window prioritization algorithm. Starting from an unordered set of test case, the algorithm generates window interaction pairs (line 2), and creates an ordered list from the given test suite (line 3). The algorithm then extracts test cases that have the same scores (line 4) and uses window/parameter interaction pairs to reassign scores for extracted test cases (lines 6-8). At the third level, the algorithm repeats the process: extracts test cases with the same scores and uses window/parameter/value interaction pairs to recalculate the scores (lines 9-13). If there are any test cases left with the same scores, random tie-breaking is applied.

The algorithm addresses the issues with fine-grained and coarse criteria and should produce more reliable results compared to single-criterion methods. We evaluate the results of the algorithm in Section 5.

5. EMPIRICAL STUDY

This study answers the following research questions:

RQ1. How effective is the fault detection rate using the multilevel coarse-to-fine-grained approach compared to the methods that use a single criterion?

RQ2. What is the difference in the fault detection rate variance between the proposed multilevel coarse-to-fine-grained method and single criterion methods?

Table 1: Object programs and associated data

Description	Word	Ssheet	Paint	OJS
Lines of code	4,893	12,791	18,376	364,290
# of classes	104	125	219	1,557
# of methods	236	579	644	13,905
# of test cases	105	268	274	109
# of faults	58	34	118	29
# of test cases with one or more faults	87	40	68	106
# of unique windows in a test suite	10	7	10	320
# of parameters in a test suite	120	148	213	132
# of parameter/value tuples in a test suite	141	186	248	467
# of sequences in a test case (max/avg)	13/4.03	9/2.54	11/2.59	74/16.75
# of faults found by a test case (max/avg)	9/3.65	7/0.24	17/1.18	5/2.27

5.1 Object Programs

We use four object applications in this study: TERP Word, TERP Paint, TERP Spreadsheet, and Open Journal Systems (OJS).

TERP Office[2] was developed at the University of Maryland using the Java programming language. It includes four applications: TERP Word, TERP Paint, TERP Spreadsheet, and TERP Calc. TERP Calc contains only two windows, which makes it an inadequate candidate for the study. The detailed description of TERP Office and the fault seeding process is provided in the previous work [8]. All faults were seeded manually and were similar to naturally occurring faults. Examples of faults include modified relational, logical, and arithmetic operators.

Open Journal Systems is a web-based journal system that was created by Public Knowledge Project. OJS test cases are user session-based test cases. User sessions were captured using web-server logs and converted for replay by our tool. The Open Journal Systems metrics were calculated using PHP Depend[3]. Multiple third-party libraries such as CodeIgniter[4] and Smarty[5] are included in OJS, which may increase a risk of additional challenges: the libraries may have compatibility issues or hidden faults and also contribute to a higher code complexity. The faults for OJS application were seeded manually by students at the University of North Texas. Categories of faults include appearance issues, incorrect behavior, broken links, and database errors. Table 1 lists the object programs and their associated data.

The original test suites were processed in order to eliminate bugs that were found by 80 or more percent of test cases and test cases that find more than 20 percent of the bugs. The removal of bugs was the first step with the removal of test cases following it. The process was repeated until there were no more such bugs and test cases present in a test suite. The number of test cases before processing for TERP Paint was 300, for TERP Spreadsheet was 282, for TERP Word was 237, and for OJS was 277. The number of bugs before processing for Paint was 182, for Spreadsheet was 79, for Word was 96, and for OJS was 67.

[2]https://www.cs.umd.edu/users/atif/TerpOffice/
[3]http://pdepend.org
[4]http://codeigniter.com
[5]http://www.smarty.net

5.2 Independent and Dependent Variables

Our independent variable is a prioritization method. We consider four control techniques and one heuristic technique:

- Control
  - Random ordering (Rnd)
  - A combinatorial prioritization method that uses pairs of windows as a criterion (Win)
  - A combinatorial prioritization method that uses inter-window pairs of parameters as a criterion (Par)
  - A combinatorial prioritization method that uses inter-window pairs of parameter/values as a criterion, also known as two-way inter-window prioritization (PV)
- Heuristic
  - A hybrid multilevel coarse-to-fine-grained prioritization method (ML)

Our dependent variable and an evaluation metric is the average percentage of faults detected (APFD) metric [16].

The APFD metric is described as follows:

$$APFD = 1 - \frac{TF_1 + TF_2 + ...TF_m}{mn} + \frac{1}{2n} \quad (1)$$

In Equation 1, n is a number of test cases in test suite TS that is ordered using some technique, m is a number of faults found by test suite TS, TF_i is the first test case in TS that finds fault i. The APFD metric represents the area under the curve on the axis where x is the number of test cases executed and y is the total number of faults found.

5.3 Experiment Process

The proposed algorithm and baselines were implemented in Python 2.7 programming language using xmltodict[6] and standard libraries. The experiment was conducted on Core i7-4770 3.4Ghz with 32GB of memory and 3TB hard drive using Ubuntu Linux 12.04 LTS operating system.

For each of the single-criterion prioritization methods, hybrid multilevel prioritization, and random ordering, the experiment was repeated 1,000 times using random tie-breaking for test cases with the same scores. As a result, 20,000 APFD values were obtained for analysis for all four object programs and five prioritization methods.

6. RESULTS

In this section, we present the results of our study and data analyses for each research question. Further, we discuss implications of our results and guidance to testers.

6.1 Data Analysis

In order to answer RQ1, we use Mann-Whitney U-test and Vargha and Delaney A_{12} effect size statistic as recommended by Arcuri and Briand [11]. The non-parametric tests were chosen because the population distributions cannot be characterized as normal and have unequal variances. Table 2 provides the results of the tests and effect sizes. Mann-Whitney U-test at a significance level of 0.05 indicates that there is a significant difference among techniques (all p-values are lower than 5.0e-03).

[6]https://github.com/martinblech/xmltodict

Table 2: Mann-Whitney U-test p-values and Vargha and Delaney A_{12} effect size measures for random ordering and single-criterion methods against hybrid multilevel prioritization.

App	ML/Rnd		ML/Win		ML/Par		ML/PV	
	p-value	A_{12}	*p-value*	A_{12}	*p-value*	A_{12}	*p-value*	A_{12}
OJS	<0.0001	0.91	<0.0001	0.79	<0.0001	0.79	<0.0001	0.95
Paint	<0.0001	1.00	<0.0001	1.00	<0.0001	1.00	<0.0001	**0.28**
Sprd	<0.0001	1.00	<0.0001	0.99	<0.0001	0.67	**0.00276**	0.54
Word	<0.0001	0.95	<0.0001	**0.38**	<0.0001	1.00	<0.0001	1.00

The Vargha-Delaney A_{12} measure range is between 0 and 1 and shows how often one technique outperforms another, given that samples are independent. For example, in Table 2, when the hybrid multilevel method (ML) is compared to the single-criterion method that utilizes window pairs (Win), the measure for OJS is 0.79, which indicates that the hybrid multilevel prioritization method (ML) outperforms the single-criterion method in 79 percent of cases, and higher values indicate stronger performance. If the A_{12} measure is less than 0.5, there is an indication that the second technique performs better than the first. If the A_{12} measure is equal to 0.5, it indicates that there is no difference between compared methods. Only two instances of the A_{12} measure show situations where the proposed method performs worse than a single-criterion method: TERP Word in the case of a single-criterion prioritization using window pairs (Win), and TERP Paint in the case of a single-criterion prioritization using inter-window parameter/value pairs (PV). In other cases, the proposed approach performs better than single-criterion methods or random ordering.

We also utilize descriptive statistics for each object program and prioritization method to determine the method with the highest mean APFD values. Table 3 provides mean, standard deviation, and maximum and minimum values for each method and each application. The hybrid multilevel prioritization algorithm (ML) outperforms all single-criterion methods in two out of four applications. The comparison to random ordering reveals that the hybrid multilevel approach produces better results in all four applications. The compar-

Table 3: Experimental results showing mean, standard deviation, maximum APFD values, and minimum APFD values.

Measure	Rnd	Win	Par	PV	ML
Open Journal System					
Mean value	0.7587	0.8155	0.8080	0.8037	**0.8187**
Standard deviation	0.0434	0.0033	0.0127	0.0090	**0.0003**
Minimum value	0.6363	0.8091	0.7683	0.7784	0.8183
Maximum value	0.8831	0.8217	0.8420	0.8281	0.8192
TERP Paint					
Mean value	0.6415	0.7119	0.8385	**0.8534**	0.8521
Standard deviation	0.0574	0.0405	0.0049	0.0017	**0.0013**
Minimum value	0.4661	0.5916	0.8249	0.8492	0.8490
Maximum value	0.8098	0.8323	0.8499	0.8585	0.8553
TERP Spreadsheet					
Mean value	0.6009	0.6450	0.7794	0.7846	**0.7859**
Standard deviation	0.0634	0.0570	0.0119	0.0104	**0.0087**
Minimum value	0.4173	0.4693	0.7462	0.7568	0.7597
Maximum value	0.7636	0.8354	0.8152	0.8139	0.8087
TERP Word					
Mean value	0.7943	**0.8431**	0.8248	0.8286	0.8394
Standard deviation	0.0296	0.0163	0.0040	0.0035	**0.0011**
Minimum value	0.6903	0.7878	0.8149	0.8194	0.8374
Maximum value	0.8772	0.8869	0.8356	0.8371	0.8414

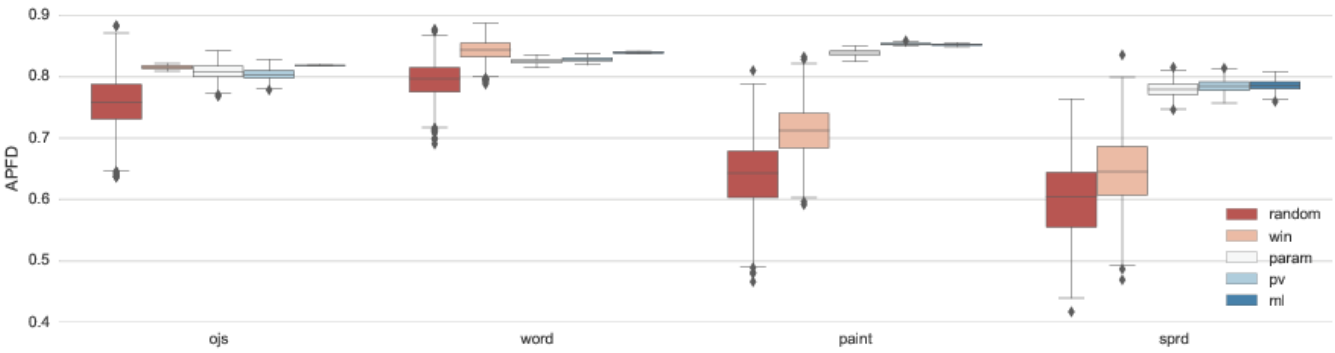

Figure 3: Box plots showing the ranges of possible APFD values for the object programs.

ison to the previously studied two-way combinatorial prioritization using parameter/values (PV) reveals that hybrid multilevel prioritization (ML) yields higher mean APFD values in three out of four applications.

In order to answer RQ2, we compared standard deviation values for the hybrid multilevel prioritization method to single-criterion methods and random ordering. The hybrid multilevel technique shows the lowest standard deviation among all methods. For instance, the standard deviation for the TERP Word application is one order of magnitude lower than that of the method with the highest APFD value. Low standard deviation values provide testers more confidence in the results.

To demonstrate our results visually, we use box plots that are shown in Figure 3. Box plots confirm previous conclusions as the proposed hybrid multilevel prioritization approach reduces variance while performing equally well or better than single-criterion approaches.

6.2 Discussion

As shown in Tables 2 and 3, the proposed approach generally produces better results with the least variance. This section explores the relationship between the results and the characteristics of the test suites of the object programs.

Given the results in Section 6.1, the prioritization methods in the control group perform differently depending on the object program. Random ordering demonstrates the worst average APFD and the highest standard deviation values among all methods, which is to be expected.

It is also expected that the single-criterion method using parameter/value pairs (PV) produces the best performance result in the control group as it has been shown previously that the PV method generally produces better results compared to other methods [8]. The PV method is also the most fine-grained method among the tested single-criterion methods. The results for TERP Paint and TERP Spreadsheet confirm previous observations: single-criterion method using parameter/value pairs1 (PV) has the highest mean APFD value among the methods in the control group.

On the contrary, in two out of four object programs, Open Journal Systems and TERP Word, the single-criterion prioritization method using window pairs (Win) produces the highest mean APFD values. To understand the reasons for such difference, we further examined the characteristics of the object programs.

In the case of OJS, the results of the single-criterion method that utilizes window pairs (Win) are not surprising considering that OJS is a web application that has a number of web pages (windows) with no parameters, in addition to having a significantly larger number of windows in a test suite among all object programs. While the standard deviation is the smallest among the control group, it is still 10 times worse than that of the proposed hybrid multilevel method (ML).

In the case of TERP Word, on the other hand, the performance of the single-criterion method using window pairs (Win) is not expected. Its performance is better than that of the proposed hybrid multilevel prioritization (ML). The answer to this phenomenon is the result of a number of factors. One of the factors is the standard deviation value: it is the second worst among all methods. Other factors that should be considered are the lowest number of test cases, the highest fault density, and the median number of windows in a test suite, which seems to indicate that combinatorial prioritization using window pairs has a high fault detection rate at the beginning, but quickly depletes available criteria, so that the remaining part of the test suite is ordered randomly.

Comparing control group methods to the heuristic, the hybrid multilevel prioritization (ML) algorithm outperforms random ordering. Hybrid multilevel prioritization shows the lowest standard deviation among the methods, which is expected as the number of tie-breaking cases decreases. The overall performance of the algorithm varies on applications. For two cases, TERP Paint and TERP Word, the hybrid multilevel approach (ML) shows the second best result.

While the combinatorial prioritization method that utilizes window pairs (Win) produces the highest average mean AFPD for TERP Word, the range of possible APFD values for the technique is almost 10 percent. Hybrid multilevel prioritization (ML), on the other hand, has the range of values less than 1 percent and has the second best mean result, which makes hybrid multilevel prioritization a more preferable choice.

The single-criterion prioritization that uses inter-window parameter/value pairs (PV) slightly outperforms the hybrid multilevel method for TERP Paint. Considering that TERP Paint has more than three times the faults of TERP Spreadsheet, and a close number of test cases, the results indicate a skew in the fault distribution between windows. As a consequence, when the hybrid multilevel method uses its first order criterion, pairs of windows (Win), the fault detection rate is lower compared to single-criterion method using parameter/value pairs (PV).

The use of test suites for TERP Spreadsheet and Open Journal System demonstrates that the hybrid multilevel prioritization technique produces the best APFD values while maintaining the lowest standard deviation, which also confirms the initial assumption that, in general, hybrid coarse-to-fine-grained multilevel prioritization techniques outperform single-criterion methods.

Overall, the proposed hybrid multilevel approach produces the lowest standard deviation, reducing testers' uncertainty in results. As indicated by Vargha and Delaney measure, it generally produces better APFD values compared to single-criterion methods or random ordering.

6.3 Threats to Validity

There are several threats to validity in this study. The applications under test and their test suites that we use may not be representative of all systems. To reduce this threat, we utilize multiple applications of different sizes: three GUI applications and one Web application. In addition, we remove so-called easy bugs, the bugs found by 80 percent or more of test suites, and so-called super test cases, test cases that find 20 percent or more of bugs to make test suites and faults more representative of real-life ones. A broader study that includes a wider variety of applications may further minimize the threat. Another threat to validity is the evaluation metric used in the study. Our choice of the metric is a widely adopted APFD metric that is used for test suite prioritization by many researchers in the field, but APFD is not the only possible measure of prioritization effectiveness. Furthermore, we did not perform a cost-benefit analysis of using the hybrid multilevel prioritization method, which may be another threat.

6.4 Guidance to Testers

The use of single criterion for test suite prioritization often leads to an increased variability of the results and decreased confidence of performance of such methods due to tie-breaking. Our recommendation for testers is to use multiple criteria for their prioritization methods in order to minimize the discussed issues. Depending on the type of test suite, testers may identify coarseness of chosen criteria and employ hybrid multilevel prioritization from coarse criteria to fine-grained. In the case of user session-based test suites, our empirical study indicates that the use of windows, parameters, and parameter/values pairs as criteria for hybrid multilevel prioritization decreases the variance and often outperforms the tested single-criterion methods.

7. CONCLUSION AND FUTURE WORK

In this paper, we demonstrated a motivational example, where the use of prioritization methods that employ a single fine-grained criterion leads to a decreased performance even when tie-breaking is not a factor. A similar example is given for the methods that employ a coarse-grained criterion. Our proposal is to use multilevel coarse-to-fine-grained prioritization that utilizes multiple criteria, which helps to decrease the variability of results and increase potential performance of prioritization. Our empirical results indicate that the proposed algorithm lowers the variability of results, in some cases significantly, and generally produces better results compared to single-criterion prioritization techniques.

In future work, we will apply the multilevel prioritization method to different types of test suites and criteria. We will conduct a comprehensive study on systematic techniques used in test suite reduction, test suite generation, and other software testing areas and how tie-breaking influences the outcomes of such techniques. Furthermore, we plan to evaluate the proposed multilevel prioritization considering cost-benefit trade-offs.

8. ACKNOWLEDGMENTS

This work was supported by National Science Foundation CAREER Award CCF-1564238 and grant CCF-1461065.

9. REFERENCES

[1] M. Wieczorek, *Systems and Software Quality : The Next Step for Industrialisation.* Heidelberg: Springer, 2014.

[2] C. Jones and O. Bonsignour, *The Economics of Software Quality*, 1st ed. Addison-Wesley Professional, 8 2011.

[3] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, P. McMinn *et al.*, "An orchestrated survey of methodologies for automated software test case generation," *Journal of Systems and Software*, vol. 86, no. 8, pp. 1978–2001, 2013.

[4] A. Causevic, D. Sundmark, and S. Punnekkat, "An industrial survey on contemporary aspects of software testing," in *International Conference on Software Testing, Verification and Validation*, 2010, pp. 393–401.

[5] 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.

[6] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test case prioritization: An empirical study," in *International Conference on Software Maintenance*, 1999, pp. 179–188.

[7] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *TSE*, vol. 28, no. 2, pp. 159–182, 2002.

[8] R. C. Bryce, S. Sampath, and A. M. Memon, "Developing a single model and test prioritization strategies for event-driven software," *TSE*, vol. 37, no. 1, pp. 48–64, 2011.

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

[10] S. Elbaum, S. Karre, and G. Rothermel, "Improving web application testing with user session data," in *Proceedings of the 25th International Conference on Software Engineering*, 2003, pp. 49–59.

[11] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Software Testing, Verification and Reliability*, vol. 24, no. 3, pp. 219–250, 2014.

[12] S. Sampath, R. Bryce, and A. M. Memon, "A uniform representation of hybrid criteria for regression testing," *TSE*, vol. 39, no. 10, pp. 1326–1344, 2013.

[13] Y. Fazlalizadeh, A. Khalilian, M. A. Azgomi, and S. Parsa, "Prioritizing test cases for resource constraint environments using historical test case performance data," in *International Conference on Computer Science and Information Technology*, 2009, pp. 190–195.

[14] R. Carlson, H. Do, and A. Denton, "A clustering approach to improving test case prioritization: An industrial case study," in *International Conference on Software Maintenance*, 2011, pp. 382–391.

[15] Z.-W. He and C.-G. Bai, "Gui test case prioritization by state-coverage criterion," in *Proceedings of the 10th International Workshop on Automation of Software Test*, 2015, pp. 18–22.

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