Exploring Better Black-Box Test Case Prioritization via Log Analysis

ZHICHAO CHEN, College of Intelligence and Computing, Tianjin University, China
JUNJIE CHEN*, College of Intelligence and Computing, Tianjin University, China
WEIJING WANG, College of Intelligence and Computing, Tianjin University, China
JIANYI ZHOU, Key Laboratory of High-Confidence Technologies (Peking University), MoE, China
MENG WANG, Department of Computer Science, University of Bristol, United Kingdom
XIANG CHEN, School of Information Science and Technology, Nantong University, China
SHAN ZHOU, Technology and Engineering Center for Space Utilization, Chinese Academy of Sciences, China
JIANMIN WANG, Technology and Engineering Center for Space Utilization, Chinese Academy of Sciences, China

Test case prioritization (TCP) has been widely studied in regression testing, which aims to optimize the execution order of test cases so as to detect more faults earlier. TCP has been divided into white-box test case prioritization (WTCP) and black-box test case prioritization (BTCP). WTCP can achieve better prioritization effectiveness by utilizing source code information, but is not applicable in many practical scenarios (where source code is unavailable, e.g., outsourced testing). BTCP has the benefit of not relying on source code information, but tends to be less effective than WTCP. That is, both WTCP and BTCP suffer from limitations in the practical use.

To improve the practicability of TCP, we aim to explore better BTCP, significantly bridging the effectiveness gap between BTCP and WTCP. In this work, instead of statically analyzing test cases themselves in existing BTCP techniques, we conduct the first study to explore whether this goal can be achieved via log analysis. Specifically, we propose to mine test logs produced during test execution to more sufficiently reflect test behaviors, and design a new BTCP framework (called LogTCP), including log pre-processing, log representation, and test case prioritization components. Based on the LogTCP framework, we instantiate seven log-based BTCP techniques by combining different log representation strategies with different prioritization strategies.

We conduct an empirical study to explore the effectiveness of LogTCP. Based on 10 diverse open-source Java projects from GitHub, we compared LogTCP with three representative BTCP techniques and four representative WTCP techniques. Our results show that all of our LogTCP techniques largely perform better

*Corresponding author.

Authors' addresses: Zhichao Chen, chenzhichao99@tju.edu.cn, College of Intelligence and Computing, Tianjin University, Tianjin, China, 300350; Junjie Chen, junjiechen@tju.edu.cn, College of Intelligence and Computing, Tianjin University, Tianjin, China, 300350; Weijing Wang, wangweijing@tju.edu.cn, College of Intelligence and Computing, Tianjin University, Tianjin, China, 300350; Jianyi Zhou, zhoujianyi@pku.edu.cn, Key Laboratory of High-Confidence Technologies (Peking University), MoE, Beijing, China, 100871; Meng Wang, meng.wang@bristol.ac.uk, Department of Computer Science, University of Bristol, Bristol, United Kingdom, BS8 1UB; Xiang Chen, xchencs@ntu.edu.cn, School of Information Science and Technology, Nantong University, Nantong, China, ; Shan Zhou, zhoushan@csu.ac.cn, Technology and Engineering Center for Space Utilization, Chinese Academy of Sciences, Beijing, China, 100094; Jianmin Wang, wangjm@csu.ac.cn, Technology and Engineering Center for Space Utilization, Chinese Academy of Sciences, Beijing, China, 100094.

1049-331X/2022/1-ART111 $15.00
https://doi.org/10.1145/1122445.1122456

than all the BTCP techniques in average fault detection, to the extent that then become competitive to the WTCP techniques. That demonstrates the great potential of logs in practical TCP.

CCS Concepts: • **Software and its engineering → Software testing and debugging**.

Additional Key Words and Phrases: Test Case Prioritization, Log Analysis, Regression Testing

ACM Reference Format:
Zhichao Chen, Junjie Chen, Weijing Wang, Jianyi Zhou, Meng Wang, Xiang Chen, Shan Zhou, and Jianmin Wang. 2022. Exploring Better Black-Box Test Case Prioritization via Log Analysis. *ACM Trans. Softw. Eng. Methodol.* 37, 4, Article 111 (January 2022), 33 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Test case prioritization (TCP) is one of the most widely-studied regression testing techniques [23, 77, 113]. It aims to schedule the execution order of test cases in order to detect more faults earlier [92]. Based on whether the source code information is utilized, TCP has been divided into two categories: white-box test case prioritization (WTCP) and black-box test case prioritization (BTCP) [47]. WTCP prioritizes test cases by conducting dynamic or static analysis on source code, including coverage-based techniques [64, 70, 92] and mutation-based techniques [35, 69, 109], whereas BTCP, in the absence of the source code information, prioritizes test cases by measuring the diversity among test cases (such as test-case text diversity) [18–20, 24, 46, 48, 60].

In the literature, a large amount of effort has been devoted to WTCP, achieving notable results in effectiveness [23, 27, 86]. However, the reliance on source code information limits its usage and imposes high costs in data collection. For example, coverage-based TCP instruments source code to obtain coverage information [35], and mutation-based TCP constructs a large number of mutants based on the source code [69]. As a result, WTCP is not applicable in many practical scenarios, in which source code is unavailable, such as outsourced testing [98]. On the other hand, BTCP has the obvious advantage of not requiring the source code, but as a result, it does have less to work with and is often considered less effective than the white-box counterpart. That is, both WTCP and BTCP suffer from limitations in their practical use.

So can we have the best of both worlds, i.e., achieving great effectiveness without relying on source code information? In other words, can we improve the effectiveness of BTCP and thus bridge the effectiveness gap between BTCP and WTCP? Existing BTCP approaches tend to focus on measuring the diversity among test cases by treating test code as text [60, 79, 108]. This is quite limited: statically inspecting test cases alone provides little information of their dynamic behaviors. Hence, the key to improving TCP effectiveness is to identify and utilize dynamic information of test executions, without having access to source code. One such source of information is the test execution logs. Produced during the running of software (for the purpose of checking software status, detecting faults, diagnosing root causes, etc), logs capture events and states of interest which shed light on software behaviors. Also importantly, logs can be collected in testing without having to access source code, since logging statements are part of the software written during the programming stage. Hence, we conjecture that utilizing logs recorded during the execution of each test case to reflect test behaviors could be a promising direction to improve BTCP effectiveness.

In this work, we conduct the first attempt to improve the effectiveness of BTCP via log analysis. Specifically, we design a new BTCP framework (called LogTCP), which includes three key steps: log pre-processing, log representation, and test case prioritization based on log vectors. Log pre-processing facilitates log analysis by abstracting log messages into log events that are structured templates designed by developers to reflect the behaviors embodied in the log messages [45]. This step also has the effect of filtering out noise and irrelevant information from the log messages. That is, the nature of logs used by LogTCP is the dynamic behavior of each test case reflected by a

sequence of log events for the corresponding test case. Then, effective features are extracted to transform a sequence of log events for a test case into a vector (log representation), ready for the prioritization task that is the last step. LogTCP is a general framework that can be instantiated with specific techniques. We propose three log representation strategies by emphasizing different kinds of features in log events, including count-based representation, ordering-based representation, and semantics-based representation. We also adapt three popular ranking strategies for prioritization based on log vectors, including the total strategy [92], additional strategy [92], and adaptive random prioritization (ARP) strategy [51]. Overall, by combining different log representation strategies with different prioritization strategies, we manage to implement seven log-based TCP techniques (the semantics-based representation strategy cannot be combined with the total or additional prioritization strategy).

To investigate the effectiveness of LogTCP, we conducted an extensive study based on 10 widely-used Java projects from GitHub, totaling 480,943 lines of source code and 17,853,105 lines of test log messages, by comparing three representative BTCP techniques and four representative WTCP techniques. In the study, we aim to address the following three research questions (RQs):

- **RQ1**: How do LogTCP techniques perform compared with existing BTCP techniques?
- **RQ2**: What is the influence of inherent factors (including both log representation strategies and prioritization strategies) in LogTCP?
- **RQ3**: What is the effectiveness gap between LogTCP and WTCP?

RQ1 aims to investigate whether LogTCP can effectively improve the effectiveness of BTCP; RQ2 aims to investigate the influence of log representation strategies and prioritization strategies on LogTCP, and then provide the suggestion of applying these LogTCP techniques in practice; RQ3 aims to investigate the degree to which the goal of possessing the advantages of both WTCP and BTCP is approached. Based on our experimental results, we find that all of our LogTCP techniques outperform all of the studied BTCP techniques (including the state-of-the-art one) in average fault detection, even achieves competitive effectiveness with the state-of-the-art WTCP technique. The results demonstrate that LogTCP is indeed able to significantly bridge the effectiveness gap between BTCP and WTCP, largely promoting the practicability of TCP. In practice, we recommend the LogTCP technique combining semantics-based or ordering-based log representation with the adaptive random prioritization strategy as the representative due to its better effectiveness than the other LogTCP techniques.

To sum up, our work makes three major contributions:

- We are the first to utilize log analysis to improve the effectiveness of BTCP, combining the advantages of both WTCP and BTCP for enhanced practicability.
- We design a log-based TCP framework (LogTCP) and implement seven specific log-based TCP techniques by proposing a series of log representation strategies and test case prioritization strategies.
- We conduct an extensive study to evaluate the effectiveness of LogTCP by comparing with both state-of-the-art BTCP and WTCP techniques, demonstrating its great potential.

2 BACKGROUND

2.1 Test Case Prioritization

As presented in the existing work [92], TCP can be formally defined to find $T' \in PT$ satisfying $\forall T'' \in PT : [f(T') \geq f(T'') \wedge T'' \neq T']$, where PT refers to a set of permutations of a test suite T, and f refers to an objective function that maps a permutation to a numerical value. Based on whether the source code information is utilized, TCP has been divided into two categories, i.e., WTCP and BTCP [47]. Here, we introduce typical WTCP and BTCP techniques, which are also

used for comparison with LogTCP in our study. We will discuss other TCP techniques as related work in Section 7.

Coverage-based TCP is the most widely-studied WTCP techniques [34, 75, 91, 92], especially the total coverage-based technique (**WTCP$_{total}$**) [92], additional coverage-based technique (**WTCP$_{additional}$**) [92], search-based coverage-based technique (**WTCP$_{search}$**) [64], and adaptive random coverage-based technique (**WTCP$_{arp}$**) [51]. We therefore use the four techniques as the representative WTCP techniques in our study. In particular, we used statement coverage as their prioritization criterion due to its effectiveness following the existing studies [23, 73, 126].

Both WTCP$_{total}$ and WTCP$_{additional}$ are greedy strategies [92]. WTCP$_{total}$ prioritizes test cases according to the descending order of the number of statements covered by test cases, while WTCP$_{additional}$ prioritizes test cases according to the number of statements that are not covered by already selected test cases but covered by unselected test cases. Although the idea is simple, WTCP$_{additional}$ has been widely recognized as a state-of-the-art technique due to its effectiveness [64, 70, 71, 122]. WTCP$_{search}$ treats all the permutations of a test suite as candidate solutions and adopts some heuristics to guide the process of searching for a better solution in terms of statement coverage. Following the existing studies [6, 119, 120], we used the same Genetic Algorithm as the one designed by Li et al. [64] as the representative in WTCP$_{search}$. It initially generates a set of permutations randomly and then produces new permutations via crossover and mutation operations in subsequent iterations. For crossover operation, two parent permutations produce two offspring permutations through crossover on a random position. For mutation operation, it randomly selects two tests and exchanges their positions for each offspring permutation. WTCP$_{arp}$ iteratively prioritizes test cases based on the diversity of their covered statements (measured by Jaccard distance). It defines various distances to determine which test case is the farthest from a set of already selected test cases in terms of covered statements. Following the existing work [23, 73], WTCP$_{arp}$ adopts the distance defined to select the test case that has the largest minimum distance with already selected test cases as the next one, since it has been demonstrated to be the most effective one among the proposed distances [51].

In the literature, three representative BTCP techniques are the string-based technique (**BTCP$_{string}$**) [60], topic-based technique (**BTCP$_{topic}$**) [108], and FAST (**BTCP$_{FAST}$**) [79]. BTCP$_{string}$ treats each test case as a string, and then adopts the adaptive random strategy (used in WTCP$_{arp}$) to prioritize test cases by considering the diversity of test-case strings. In our study, it uses the Levenshtein distance to measure the distance between strings following the existing study [47]. BTCP$_{topic}$ treats each test case as text and adopts the Latent Dirichlet Allocation (LDA) algorithm [9] to extract topics (which can approximate the functionality of each test case by mining hidden semantics) from the text. In this way, a test case can be represented as a topic vector, in which each element refers to the proportion of words in the test-case text that come from the corresponding topic. Then, BTCP$_{topic}$ uses the adaptive random strategy to prioritize test cases by considering the diversity of topic vectors (measured by Manhattan distance). BTCP$_{FAST}$, the state-of-the-art BTCP technique, also treats each test case as a string, and adopts the data mining algorithms (i.e., *minhashing* and *locality-sensitive hashing* algorithms [88]) to speed up the process of finding diverse test cases after transforming each string to a k-shingle (the set of its substrings of length k). In BTCP$_{FAST}$, it uses a function to balance efficiency and accuracy and here we use the *all* function as the representative due to its effectiveness demonstrated by its experiment. Also, FAST has been applied to improve the *efficiency* of coverage-based WTCP and the existing study has shown that their prioritization *effectiveness* is not significantly affected [79]. Hence, we did not study FAST on WTCP since our study focuses on the *effectiveness* comparison.

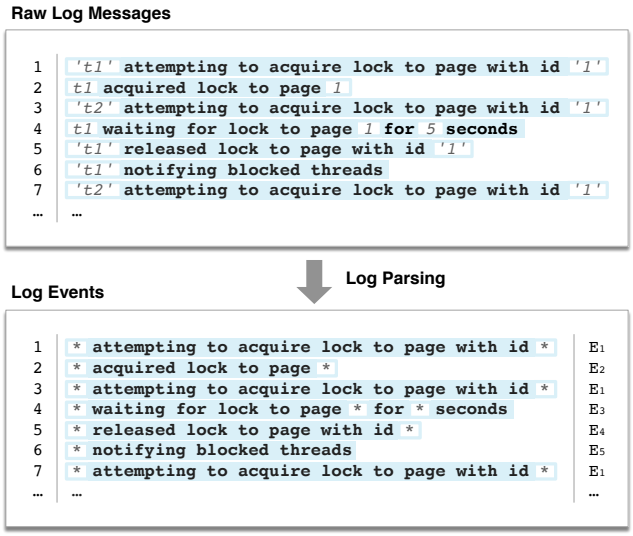

Fig. 1. An example of raw log messages and log events obtained by log parsing

2.2 Log Terminology

Logs contain abundant information reflecting the running status of a software system. In practice, developers tend to examine logs to check software behaviors, detect faults, and diagnose root causes. Here, for ease of understanding, we use an example (shown in Figure 1) to introduce log terminology used in this paper. This example is a part of test logs from the open-source Java project *Wicket*[1]. From Figure 1, a **log message** is a raw unstructured sentence generated during test execution. In this example, *'t1'* and *'t2'* represent the names of the processes that try to perform some operations on the page's lock. A log message (e.g., *'t1' attempting to acquire lock to page with id '1'*) consists of a **log event** (e.g., * attempting to acquire lock to page with id *) and **log parameters** (e.g., *'t1'* and *'1'*). A log event is the template of a log message that is written by developers during the programming stage, while log parameters are the variable part in a log message, which records some system attributes (e.g., path and id). In log analysis, **log parsing** tends to be used to extract log events from log messages. Each unique log event can be assigned with a unique ID (e.g., E1 and E2 in Figure 1) in order to facilitate the follow-up log analysis, and we can find that different log messages may have the same log event. After executing a test case, a series of log messages can be produced, which is called a **log-message sequence**. Similarly, a series of log events extracted from a log-message sequence is called a **log-event sequence**, which records a specific execution flow through the test case and can help reflect the dynamic behaviors of the test case to some degree.

3 LOG-BASED TEST CASE PRIORITIZATION

3.1 Overview

To improve the effectiveness of BTCP, we design a general log-based BTCP framework, called LogTCP, which includes three key components: log pre-processing, log representation, and test case prioritization. Figure 2 shows the architecture of LogTCP.

[1]https://wicket.apache.org/ (Accessed on: 4 January 2022)

In LogTCP, there are three challenges that have to be handled. First, logs are a kind of semi-structured natural language text and thus analyzing them is non-trivial. In particular, not all the contents in logs could contribute to understanding test behaviors, e.g., some log parameters. Hence, it is necessary to extract useful information from logs and transform them into an easy-to-analyze form. To solve this problem, the log pre-processing component (to be presented in Section 3.2) conducts log parsing to extract log-event sequences. On one hand, a log-event sequence can effectively reflect what a test case does in its execution; on the other hand, log events are structured information, which can facilitate log analysis.

The second challenge is how to represent the log-event sequence of each test case for the prioritization task. Since each log event could reflect one action conducted by a test case in its execution, the counts, ordering, and semantics of log events may embody the test behaviors of each test case. Hence, we design three strategies in the log representation component (to be presented in Section 3.3), i.e., count-based representation, ordering-based representation, and semantics-based representation, to represent a sequence of log events produced during the execution of a test case as a log vector.

The third challenge is how to prioritize test cases based on their corresponding log vectors. Here, the component of test case prioritization (to be presented in Section 3.4) adapted three widely-studied prioritization strategies in WTCP to log-based test case prioritization, including the total strategy, additional strategy, and adaptive random prioritization strategy.

Please note that our LogTCP framework is general, and thus future advances in log representation strategies or test case prioritization strategies can be integrated into LogTCP. In current LogTCP, by combining the three log representation strategies with the three test case prioritization strategies, we implemented seven log-based BTCP techniques since the semantics-based representation strategy cannot be combined with the total or additional prioritization strategy.

3.2 Log Pre-processing

During the execution of a test case, a sequence of log messages could be produced to record the test behaviors of the test case. The generation of log messages tend to be controlled by the logging levels provided by the used logging frameworks (such as Log4j and Logback). As shown in Figure 2(a), raw log messages are unstructured data and contain variable log parameters, which could hinder automatic log analysis [45]. Therefore, in this component LogTCP first conducts log parsing to extract the log-event sequence from a log-message sequence in order to filter out some useless information and facilitate follow-up log analysis based on structured log events. That is, the minimal information in logs required by LogTCP is just the log events, which could be different under different logging levels. In particular, we carefully investigated the influence of logging levels on the effectiveness of LogTCP in Section 6.1. Here, LogTCP adopts one of the most widely-used log parsing tools, i.e., Drain3 [44], since it has been demonstrated to be very efficient and accurate in the existing study [129]. Specifically, Drain3 employs a fixed-depth parse tree to guide the log-parsing process by designing several parsing rules.

Although each log event is structured, it actually can be treated as a sentence in natural language that is programmed by a developer. Typically, there are non-character tokens (e.g., delimiters, operators, and punctuation marks) and composite tokens that are concatenations of words (e.g., *NullPointerException*). Hence, to facilitate the understanding of each log event (especially in semantics-based representation), it is also necessary to conduct natural language pre-processing on these log events. Specifically, LogTCP first removes non-character tokens and stop words from each log event and then splits composite tokens into individual words using the Camel Case heuristics [30].

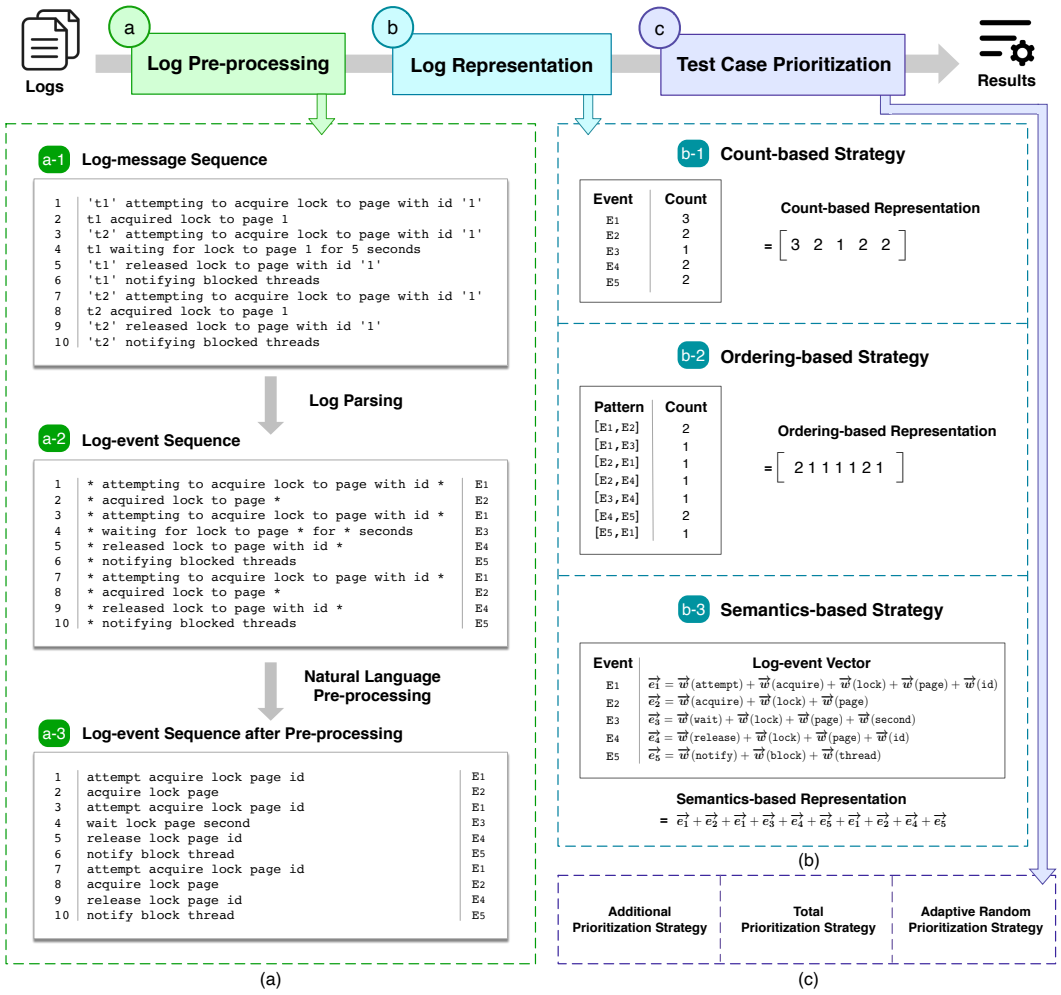

Fig. 2. Workflow of LogTCP

3.3 Log Representation

Based on different features in log-event sequences of test cases, we design three log representation strategies in LogTCP. In each strategy, the log-event sequence of each test case can be represented as a vector for the follow-up prioritization task. For ease of understanding, we also use an example (shown in Figure 2(b)) to help illustrate each strategy.

3.3.1 Count-based Representation. A log event can reflect an action that is performed by a test case in its execution. Intuitively, the categories of log events and the number of each category of log events in a log-event sequence could help model the test behaviors of a test case. For instance, if there are 10 categories of log events in a log-event sequence, one category of log events (denoted as E_1) occurs 10 times but the remaining categories occur only once respectively, which may mean that this test case focuses on performing the action of E_1 in order to test a certain functionality sufficiently. Therefore, the count-based representation strategy counts the number of each category of log events in a log-event sequence in order to transform the log-event sequence into a vector.

More specifically, this strategy transforms a log-event sequence of a test case into a n-dimension vector, denoted as $C_i = \{c_{i1}, c_{i2}, \ldots, c_{in}\}$, where n refers to the total number of log-event categories occurring in this project under test and c_{ij} refers to the number of occurring times of the j^{th} log event in the log-event sequence of the i^{th} test case. For the log-event sequence example shown in Figure 2(a) (assuming $n = 5$), it is represented as a 5-dimension vector ($[3, 2, 1, 2, 2]$ shown in Figure 2(b-1)) by counting the number of E1, E2, E3, E4, and E5 respectively.

3.3.2 Ordering-based Representation. This strategy considers the ordering of log events to help represent the test behaviors of a test case, instead of each individual log event used in the count-based representation strategy. The insight behind this strategy is that the ordering of log events in a log-event sequence can reflect the execution logic of a test case to some degree. That is, different contexts (i.e., the adjacent log events) of a log event in a log-event sequence may reflect its different testing purposes. To extract the ordering features of log events, we incorporate the N-Gram model [13], which is widely used in the area of natural language processing to operate the contents of text through a sliding window with a size of N. In our scenario, this strategy uses the N-Gram model to extract a set of log-event sub-sequences from a log-event sequence with the size of N. By regarding a log-event sub-sequence as a pattern, it then counts the number of occurring times of each pattern in a log-event sequence. In this way, the log-event sequence of a test case can be transformed into a m-dimension vector, denoted as $O_i = \{o_{i1}, o_{i2}, \ldots, o_{im}\}$, where m refers to the total number of log-event sub-sequence patterns occurring in this project under test and o_{ij} refers to the number of occurring times of the j^{th} pattern in the log-event sequence of the i^{th} test case. For the log-event sequence example shown in Figure 2(a) (assuming $N = 2$ and $m = 7$), it is represented as a 7-dimension vector $[2, 1, 1, 1, 1, 2, 1]$ shown in Figure 2(b-2) by counting the number of occurring times of each log-event sub-sequence pattern (e.g., the pattern [E1,E2] occurs twice) in this log-event sequence.

3.3.3 Semantics-based Representation. As presented in Section 3.2, each log event can be regarded as a sentence in natural language, and thus each log event as well as each log-event sequence have their own semantics. If the log-event sequences of two test cases have similar semantics, it is likely for them to share similar test behaviors. With this insight, this strategy extracts semantic features from the log-event sequence of a test case to facilitate the representation of its test behaviors.

Specifically, following the existing work on log analysis [45, 66, 117, 123], this strategy first transforms each word in a log event into a d-dimension vector by extracting the semantic information from the word through word embedding. Here, it conducts the word-embedding task through a pre-trained word2vec model using the FastText algorithm [10], which can effectively capture the intrinsic relationship among words in natural language. After obtaining the d-dimension word vector for each word in a log event (denoted as $W_{ij} = \{w_{ij}^1, w_{ij}^2, \ldots, w_{ij}^d\}$, where W_{ij} refers to the word vector of the j^{th} word in the i^{th} log event of the log-event sequence), this strategy further aggregates all word vectors in the log event to a log-event vector, denoted as $E_i = \{e_{i1}, e_{i2}, \ldots, e_{id}\}$ where $e_{ik} = \sum_{j=1}^{r} w_{ij}^k$ $(1 \leq k \leq d)$ and r is the total number of words in the i^{th} log event. Finally, it obtains the semantic vector of the log-event sequence by aggregating all log-event vectors, denoted as $S = \{s_1, s_2, \ldots, s_d\}$ where $s_g = \sum_{h=1}^{t} e_{hg}$ $(1 \leq g \leq d)$ and t is the total number of log events in the log-event sequence. Here, we adopt the summation method for vector aggregation in order to incorporate the length information of each log event and the log-event sequence. For the log-event sequence example shown in Figure 2(a), it can be represented as a d-dimension vector shown in Figure 2(b-3) by first obtaining each word vector and then aggregating them through summation.

Note: Our work aims to conduct the first exploration on log-based TCP, and in this work we design the above three strategies to model test behaviors based on logs. Although the three strategies have

considered three different kinds of features from test logs, it is likely to have other kinds of features that may also help model test behaviors. Moreover, even for the three kinds of features, there could be also other methods that can represent them as vectors. In fact, the main contribution of our work lies in firstly exploring the potential of log-based TCP, and thus we take the integration of more advanced representation strategies into our LogTCP framework as future work.

3.4 Test Case Prioritization

Based on the set of log vectors produced by a log representation strategy, we adapt three widely-studied prioritization strategies in this component, in order to produce the prioritization result of test cases.

3.4.1 Total and Additional Prioritization Strategies. The total and additional strategies are originally proposed for coverage-based test case prioritization (as presented in Section 2.1). In our scenario, we adapt the total and additional strategies based on the *coverage of log-event categories* for the log vectors produced by the count-based representation strategy or the *coverage of the log-event sub-sequence patterns* for the log vectors produced by the ordering-based representation strategy, instead of program elements used in coverage-based TCP. Please note that, we cannot apply the total and additional strategies to the log vectors produced by the semantics-based representation strategy since this representation strategy does not involve the concept of coverage.

3.4.2 Adaptive Random Prioritization Strategy. The adaptive random prioritization (ARP) strategy is originally proposed to prioritize test cases based on code coverage diversity as presented in Section 2.1. In our scenario, we adapt it based on the diversity of log vectors. That is, it defines the distance between log vectors to determine which test case should be selected next during prioritization. Specifically, it iteratively selects the test case that has the largest minimum distance with the already prioritized test cases following the existing study [41, 72, 73]. Here, we study three distances to measure the diversity of log vectors, including *Manhattan* Distance, *Euclidean* Distance, and *Cosine* Distance as shown in Formula 1, Formula 2, and Formula 3, respectively.

$$d_{manhattan}(x, y) = \sum_{i=1}^{N} |x_i - y_i| \tag{1}$$

$$d_{euclidean}(x, y) = \sqrt{\sum_{i=1}^{N} (x_i - y_i)^2} \tag{2}$$

$$d_{cosine}(x, y) = 1 - \frac{\sum_{i=1}^{N} x_i \times y_i}{\sqrt{\sum_{i=1}^{N} x_i^2} \times \sqrt{\sum_{i=1}^{N} y_i^2}} \tag{3}$$

where x and y refer to two N-dimension log vectors. In particular, before measuring the distance between log vectors, it is required to normalize these vectors in order to adjust the feature values to a common scale (i.e., the interval $[0, 1]$) for more precise diversity measuring. Following the existing work, it adopts the widely-used min-max normalization method [40]. The original ARP strategy randomly selects the first test case, which could lead to unstable performance. Hence, to reduce the randomness of the ARP strategy, our adapted ARP strategy selects the test case with the largest number of log-event categories as the first one in the prioritization result. The ARP strategy is applicable to the log vectors produced by any of the three log representation strategies.

Note: Similar to the discussion on log representation strategies, it is also possible to have other prioritization strategies, and we will integrate more advanced test case prioritization strategies

into our LogTCP framework in the future. Our work puts more efforts into exploring the potential of mining test logs for better BTCP.

4 EVALUATION DESIGN

In this study, our main goal is to investigate whether LogTCP can effectively improve the effectiveness of BTCP compared with the state-of-the-art BTCP. Also, we investigated the influence of different log representation strategies and different test case prioritization strategies on the effectiveness of LogTCP, in order to suggest how to apply and further improve our log-based techniques in practice. Finally, we compared our log-based techniques with the state-of-the-art WTCP to investigate the effectiveness gap between LogTCP and WTCP, which is helpful to answer whether we have approached our expectation (i.e., achieving great effectiveness without relying on source code information). The detailed RQs have been presented in Section 1. Here, we present our study design in detail.

4.1 Subjects and Faults

In the study, we used 10 open-source Java projects from GitHub as subjects, which are widely-used in the existing studies on log analysis [16, 42, 61] or test case prioritization [23, 73, 126]. All these subjects are built with the Maven framework[2], manage test cases based on the Junit framework[3], and produce logs based on the Log4j[4] or Logback[5] (a successor to Log4j) library (we set the logging level to *ALL* for all the subjects in our study and will discuss the influence of different logging levels on the effectiveness of LogTCP in Section 6.1). Table 1 presents the basic information of these subjects, in which each column represents the project ID, the project name, the commit ID of the project, the number of lines of source code (SLOC), the number of lines of test code (TLOC), the number of lines of test log messages produced during test execution (LLOC), the number of test classes, and the number of test methods, respectively. In total, there are 480,391 SLOC, 387,759 TLOC, and 17,853,105 LLOC. In particular, these subjects have great diversity, e.g., involving diverse domains, having different functionalities, and having different scales. In particular, for each subject we ran each test case several times (i.e., 10 times in our study) for identifying and removing flaky tests following the existing work [7, 58, 78, 102].

Following the existing studies on test case prioritization [23, 73, 113], we used mutation faults to evaluate the effectiveness of the studied TCP techniques since the existing studies [3, 21, 22, 54, 70, 109] have demonstrated that mutation faults are suitable for software testing experimentation. Moreover, it is very challenging to collect a large number of real regression faults for evaluation [23, 70]. Indeed, some TCP studies have used the real faults provided by Defects4J[53], but this benchmark does not provide logs and thus we cannot use it in our study. We also discussed this kind of potential threat from mutation faults in Section 6.5.

Specifically, for each subject we first adopted PIT[6], one of the most widely-used mutation tools, to generate mutant faults. Here, we used all the mutation operators provided by PIT to generate mutation faults. For each mutation fault, we ran each test case on it, and determined that this mutation fault is killed by a test case if the test case produces different testing results between the original project and the mutated version. According to the conclusion from the existing study of investigating the threats of mutant faults [83], we then filtered out all the duplicate mutation faults since they could exaggerate the effectiveness of TCP techniques in terms of fault detection. More

[2]https://maven.apache.org/ (Accessed on: 4 January 2022).
[3]https://junit.org/junit4/ (Accessed on: 4 January 2022).
[4]https://logging.apache.org/log4j (Accessed on: 4 January 2022).
[5]http://logback.qos.ch/ (Accessed on: 4 January 2022).
[6]http://pitest.org (Accessed on: 4 January 2022).

Table 1. Basic information of subjects

ID	Project	Commit ID	SLOC	TLOC	LLOC	#Test Class	#Test Method
1	ActiveMQ-amqp	1f3ccad9	8,532	30,561	2,825,979	85	1,698
2	Airavata-registry-core	efd6bd25	30,869	6,069	35,800	32	64
3	Blueflood-http	b952e857	3,241	3,428	240	25	177
4	Dubbo-config-spring	9783ef06	11,818	13,955	12,388	22	49
5	Flume-ng-core	d17f0a46	23,563	20,696	594	31	129
6	Kylin-core-metadata	2fb07e6b	31,915	6,899	675	54	176
7	ORCID-Source-core	68cff155	135,137	154,500	9,905,244	240	1,791
8	Shiro-core	b637c467	28,894	8,602	5,701	64	294
9	Webdrivermanager	e1453c4c	6,583	2,428	848,878	83	212
10	Wicket-core	34f78c85	200,391	140,621	4,217,606	455	2,083
Total			480,943	387,759	17,853,105	1,091	6,673

Table 2. Seven Log-based TCP techniques

Technique	Log representation strategy	Prioritization strategy
$\text{LogTCP}^{total}_{count}$	count-based	total
$\text{LogTCP}^{additional}_{count}$	count-based	additional
$\text{LogTCP}^{arp}_{count}$	count-based	ARP
$\text{LogTCP}^{total}_{ordering}$	ordering-based	total
$\text{LogTCP}^{additional}_{ordering}$	ordering-based	additional
$\text{LogTCP}^{arp}_{ordering}$	ordering-based	ARP
$\text{LogTCP}^{arp}_{semantics}$	semantics-based	ARP

specifically, if two mutation faults can be killed by the same set of test cases, they are regarded as duplicate mutation faults and only one of them is kept as the representative. Also, we removed all the live mutation faults that cannot be killed by any test cases. Finally, following the practice of many existing studies [23, 70, 73], we randomly selected 500 mutation faults from the set of remaining mutation faults, and constructed 100 mutation groups, each of which contains 5 randomly selected mutation faults. That is, we constructed 100 faulty versions for each subject and each version contains 5 mutation faults. If the total number of mutant faults after filtering is less than 500, the number of mutation groups is also less than 100.

4.2 Studied TCP Techniques

4.2.1 Compared Techniques. As presented in Section 2.1, we compared LogTCP with four representative WTCP techniques and three representative BTCP techniques. The four WTCP techniques are WTCP_{total}, $\text{WTCP}_{additional}$ (a state-of-the-art WTCP technique), WTCP_{search}, and WTCP_{arp}, and the three BTCP techniques are BTCP_{string}, BTCP_{topic}, and BTCP_{FAST} (the state-of-the-art BTCP technique).

4.2.2 Our Log-based Techniques. As presented in Section 3, we constructed seven log-based TCP techniques based on our LogTCP framework by combining different log representation strategies with different prioritization strategies respectively. For ease of presentation, we listed all the seven techniques in Table 2, where the three columns present the name of a log-based TCP technique, the log representation strategy used by the technique, and the prioritization strategy used by

the technique. We took the last row as an example for further explanation: the $\text{LogTCP}^{arp}_{semantics}$ technique adopts the semantics-based representation strategy for log representation and adopts the ARP strategy for test case prioritization. By comparing these techniques, we can investigate the influence of different log representation strategies and different prioritization strategies.

Besides, for the ARP strategy in LogTCP, we also studied the influence of different log-vector distances, including Manhattan distance, Euclidean distance, and Cosine distance (we did not list this independent variable in Table 2 since we investigated it for only the ARP strategy). Except for studying the influence of different log-vector distances in RQ2, we used *Euclidean* distance as the default one in our ARP strategy in LogTCP. The reasons are twofold: (1) Our study (as shown in Finding 3) demonstrates that *Euclidean* distance performs better than both Manhattan distance and Cosine distance for our ARP strategy in LogTCP. (2) The existing studies [14, 25, 125] also recommend to use Euclidean distance in ARP-based test case prioritization.

4.3 Measurements

In the study, we used two metrics, i.e., APFD and RAUC-s, to measure the effectiveness of each TCP technique following the existing work [12, 23, 73, 113, 114].

APFD: Average Percentage of Faults Detected (APFD) [47, 70, 73, 90, 91] is the most widely-used metric to measure TCP effectiveness. The calculation of APFD is shown in Formula 4:

$$APFD = 1 - \frac{TF_1 + TF_2 + ... + TF_m}{n * m} + \frac{1}{2n} \tag{4}$$

where n is the total number of test cases to be prioritized, m is the total number of detected faults by these test cases, and TF_i refers to the rank of the first test case in the prioritized result that detects the i^{th} fault. Larger APFD values mean better prioritization effectiveness.

RAUC-s: Actually, APFD measures TCP effectiveness from the angle of executing the entire prioritized test suite. However, due to the testing time limitation in practice, the entire prioritized test suite may be not always executed completely [76, 104]. Therefore, it is also necessary to evaluate TCP effectiveness when only the first s test cases in the prioritization result can be executed within the limited testing time. Following the existing work [114], we used RAUC-s to achieve this measurement goal, which measures the degree that the prioritization result of a TCP technique approaches the ideal prioritization result within the first s test cases. Specifically, it transforms the prioritization result into a plot, where the x-axis represents the number of test cases prioritized by a TCP technique and the y-axis represents the number of faults detected. Then, it calculates the ratio of the area under the curve for the TCP technique to the area under the curve of the ideal prioritization for the first s test cases. Here, according to the relationship between each test case and each used mutation fault, the ideal prioritization ranks test cases by iteratively selecting the test case that kills the largest number of mutation faults that are not killed by the already selected test cases. In our study, we consider s to be 25%, 50%, 75% of the total number of test cases, and denote them as **RAUC-25%**, **RAUC-50%**, and **RAUC-75%**, respectively. We also presented the prioritization effectiveness on all the test cases for each subject, denoted as **RAUC-100%**. The larger the RAUC-s value is, the better the TCP technique performs.

4.4 Implementation and Configurations

In LogTCP, we adopted Drain3[7] with its default settings to perform log parsing, PIT with all mutation operators to generate mutation faults, and VS Code Counter[8] to measure SLOC and TLOC

[7]https://github.com/IBM/Drain3 (Accessed on: 4 January 2022).

[8]https://marketplace.visualstudio.com/items?itemName=uctakeoff.vscode-counter (Accessed on: 4 January 2022).

for each subject. For the studied WTCP techniques, we adopted OpenClover[9] to collect coverage information. In our ordering-based log representation strategy, we set N to 2 by balancing the TCP effectiveness and efficiency. In our semantics-based log representation strategy, we set the word-vector dimension d to 50 and set the hyper-parameters in the FastText algorithm through grid search (e.g., adopting the *skipgram* model and setting the max length of word ngrams to 3). Regarding the hyper-parameters in WTCP_{search}, we did not set them specially for each subject, but used the uniform configuration for all the subjects same as the existing work [23, 64, 70, 126]. The effectiveness of the uniform configuration has been demonstrated by these existing studies [23, 64, 126]. Specifically, we set the population size to 100, the number of iterations to 300, and the probabilities for crossover and mutation operations to 0.8 and 0.1 in the Genetic Algorithm. In BTCP_{topic}, we set the number of topics to *N/2.5* following the existing work [108], where N is the number of test cases. All the settings have been reported in our project homepage[10].

Our LogTCP framework and experimental scripts are mainly implemented in Python. We have released all our implementations and experimental data in our project homepage, to promote future research and practical use. In particular, we design a series of APIs in our LogTCP framework, in order to facilitate its extension by integrating more advanced log representation strategies and prioritization strategies in the future. To reduce the influence of randomness, we repeated all the TCP techniques involving randomness 5 times and calculated the average results in our study[11]. In the study, we prioritized test cases at the test-class level following the existing studies [57, 104]. This is because different test classes have to be frequently switched/loaded for running these test methods in order when applying test-method-level TCP in practice, which can incur extra non-negligible costs.

Our study was conducted on a workstation with 20-core Intel Xeon E5-2640 CPU(2.4GHz), 126G memory, and Ubuntu 18.04.5 LTS.

5 RESULTS AND ANALYSIS

5.1 RQ1: LogTCP v.s. Existing BTCP Techniques

To investigate whether LogTCP can improve the effectiveness of BTCP, we compared our LogTCP techniques with three representative BTCP techniques. Table 3 shows the comparison results in terms of average APFD and average RAUC-s across all the faulty versions for each subject. In this table, we marked the best result as the **bold** value for each subject in terms of each metric, and the last column shows the average result across all the subjects in terms of each metric.

From Table 3, our LogTCP techniques always occupy the best results on all the subjects in terms of all the metrics. For example, in terms of APFD, $\text{LogTCP}^{arp}_{semantics}$ achieves the best results on five subjects, $\text{LogTCP}^{additional}_{count}$, $\text{LogTCP}^{additional}_{ordering}$, and $\text{LogTCP}^{arp}_{ordering}$ achieve the best results on two subjects respectively, and $\text{LogTCP}^{arp}_{count}$ achieves the best result on one subject, while the BTCP techniques do not perform the best on any subjects. In terms of average APFD across all the subjects, our LogTCP techniques achieve 0.7714~0.7969, while the compared BTCP techniques achieve 0.6783~0.7014. Also, our LogTCP techniques and the compared BTCP techniques achieve 0.7604~0.8291 and 0.4386~0.5453 in term of average RAUC-25%, 0.8256~0.8715 and 0.6065~0.6650 in term of average RAUC-50%, 0.8639~0.8936 and 0.7152~0.7501 in term of average RAUC-75%, 0.8944~0.9161 and 0.7851~0.8104 in term of average RAUC-100% respectively. We can find that in terms of all these average metrics, all the LogTCP techniques perform better than all the studied

[9]http://openclover.org/ (Accessed on: 4 January 2022).
[10]https://github.com/VikingStudyHard/LogTCP. (Accessed on: 4 January 2022)
[11]Following the released implementation of BTCP_{FAST} by the existing work [79], we repeated BTCP_{FAST} 50 times and calculated the average result.

Table 3. Comparison between LogTCP and existing BTCP techniques

Metrics	Approach	1	2	3	4	5	6	7	8	9	10	Average
APFD	$BTCP_{string}$	0.7442	0.6561	0.5373	0.6361	0.5630	0.6031	0.6616	0.7980	0.6778	0.9060	0.6783
	$BTCP_{topic}$	0.6966	0.6772	0.5649	0.6676	0.5764	0.6707	0.6250	0.7524	0.6968	0.8620	0.6790
	$BTCP_{FAST}$	0.7223	0.7016	0.5364	0.6888	0.6213	0.7408	0.6005	0.7809	0.7296	0.8915	0.7014
	$LogTCP^{total}_{count}$	0.7208	0.7004	0.7151	0.8282	0.6965	0.7821	0.7052	0.8806	0.7651	0.9198	0.7714
	$LogTCP^{additional}_{count}$	0.7362	**0.7576**	0.7302	0.8312	**0.7005**	0.7921	0.7145	0.8540	0.7857	0.9293	0.7831
	$LogTCP^{arp}_{count}$	0.7503	0.7478	0.7284	0.8476	**0.7005**	0.7852	0.7129	0.8823	0.7878	0.9364	0.7879
	$LogTCP^{total}_{ordering}$	0.7194	0.7076	0.7169	0.8336	0.6990	0.7886	0.7045	0.8734	0.7725	0.9218	0.7737
	$LogTCP^{additional}_{ordering}$	0.7397	0.7499	0.7151	0.8306	0.6896	0.8030	**0.7260**	0.8809	0.7884	**0.9385**	0.7862
	$LogTCP^{arp}_{ordering}$	0.7557	0.7466	0.7267	0.8433	0.6950	0.8084	0.7135	**0.8834**	0.7841	**0.9385**	0.7895
	$LogTCP^{arp}_{semantics}$	**0.7740**	0.7496	**0.7444**	**0.8633**	0.6931	**0.8187**	0.7136	0.8764	**0.8132**	0.9230	**0.7969**
RAUC-25%	$BTCP_{string}$	0.5660	0.5648	0.3974	0.1617	0.3584	0.2676	0.3452	0.5749	0.3834	0.7668	0.4386
	$BTCP_{topic}$	0.4221	0.6527	0.4038	0.3078	0.3699	0.3692	0.2793	0.5953	0.4417	0.6562	0.4498
	$BTCP_{FAST}$	0.5582	0.5491	0.3958	0.4174	0.5313	0.5959	0.3167	0.6634	0.6897	0.7358	0.5453
	$LogTCP^{total}_{count}$	0.5205	0.5113	0.8814	0.8866	**0.9361**	0.7534	0.6111	**0.9358**	0.7208	0.8471	0.7604
	$LogTCP^{additional}_{count}$	0.6316	**0.7438**	**0.9583**	0.9005	**0.9361**	0.7835	0.6882	0.8626	0.7880	0.8556	0.8148
	$LogTCP^{arp}_{count}$	0.6215	0.7395	0.9199	0.9252	**0.9361**	0.7803	**0.7421**	0.9268	0.7898	0.8578	0.8239
	$LogTCP^{total}_{ordering}$	0.5526	0.5241	0.9006	0.9127	0.9292	0.7751	0.6127	0.9291	0.7208	0.8505	0.7707
	$LogTCP^{additional}_{ordering}$	0.6807	0.7224	0.9231	0.8988	0.9064	**0.8292**	0.7043	0.9280	0.7650	**0.8671**	0.8225
	$LogTCP^{arp}_{ordering}$	**0.7094**	0.7353	0.9135	0.9200	0.9064	0.8201	0.7116	0.9327	0.7580	0.8670	0.8274
	$LogTCP^{arp}_{semantics}$	0.6919	0.7402	0.8944	**0.9704**	0.8493	0.8257	0.6789	0.9073	**0.9149**	0.8184	**0.8291**
RAUC-50%	$BTCP_{string}$	0.6948	0.6304	0.5027	0.4413	0.5336	0.4401	0.5653	0.7639	0.6431	0.8496	0.6065
	$BTCP_{topic}$	0.6096	0.6794	0.5396	0.5697	0.5941	0.5681	0.4908	0.6889	0.6629	0.7723	0.6175
	$BTCP_{FAST}$	0.6758	0.6951	0.4756	0.5697	0.6818	0.7434	0.4713	0.7561	0.7574	0.8242	0.6650
	$LogTCP^{total}_{count}$	0.6646	0.7033	0.8989	0.8927	0.9126	0.8302	0.7071	0.9436	0.8194	0.8836	0.8256
	$LogTCP^{additional}_{count}$	0.7094	**0.8495**	**0.9303**	0.8934	**0.9160**	0.8525	0.7312	0.8798	0.8775	0.8981	0.8538
	$LogTCP^{arp}_{count}$	0.7221	0.8416	0.9180	0.9243	0.9143	0.8424	**0.7536**	0.9497	0.8732	0.9069	0.8646
	$LogTCP^{total}_{ordering}$	0.6591	0.7258	0.8989	0.9053	0.9092	0.8367	0.7052	0.9405	0.8279	0.8872	0.8296
	$LogTCP^{additional}_{ordering}$	0.7132	0.8377	0.8948	0.8998	0.8958	**0.8760**	0.7457	0.9501	0.8661	0.9099	0.8589
	$LogTCP^{arp}_{ordering}$	0.7482	0.8430	0.9180	0.9180	0.8958	0.8701	0.7419	**0.9528**	0.8633	**0.9109**	0.8662
	$LogTCP^{arp}_{semantics}$	**0.7813**	0.8184	0.8828	**0.9716**	0.8891	0.8726	0.7342	0.9382	**0.9447**	0.8818	**0.8715**
RAUC-75%	$BTCP_{string}$	0.7693	0.7359	0.5915	0.6329	0.6745	0.5962	0.6820	0.8301	0.7479	0.8912	0.7152
	$BTCP_{topic}$	0.7041	0.7587	0.6381	0.6843	0.7047	0.7060	0.6294	0.7633	0.7678	0.8387	0.7195
	$BTCP_{FAST}$	0.7494	0.7974	0.5529	0.7069	0.7853	0.8243	0.5886	0.8063	0.8169	0.8729	0.7501
	$LogTCP^{total}_{count}$	0.7467	0.8044	0.8897	0.9087	0.9243	0.8838	0.7557	0.9441	0.8710	0.9102	0.8639
	$LogTCP^{additional}_{count}$	0.7699	**0.9023**	**0.9167**	0.9129	0.9247	0.9011	0.7687	0.9045	0.9053	0.9226	0.8829
	$LogTCP^{arp}_{count}$	0.7858	0.8876	0.9085	0.9352	**0.9372**	0.8910	0.7764	**0.9498**	0.9032	0.9327	0.8907
	$LogTCP^{total}_{ordering}$	0.7435	0.8170	0.8930	0.9171	0.9291	0.8877	0.7557	0.9399	0.8773	0.9130	0.8673
	$LogTCP^{additional}_{ordering}$	0.7754	0.8887	0.8881	0.9124	0.9123	**0.9176**	**0.7875**	0.9479	0.9003	0.9350	0.8865
	$LogTCP^{arp}_{ordering}$	0.7935	0.8882	0.9085	0.9296	0.9180	0.9145	0.7699	0.9494	0.8986	**0.9354**	0.8906
	$LogTCP^{arp}_{semantics}$	**0.8216**	0.8739	0.8808	**0.9593**	0.9195	0.9093	0.7708	0.9393	**0.9469**	0.9146	**0.8936**
RAUC-100%	$BTCP_{string}$	0.8293	0.8066	0.6818	0.7210	0.7624	0.7012	0.7612	0.8682	0.8016	0.9177	0.7851
	$BTCP_{topic}$	0.7760	0.8263	0.7204	0.7553	0.7805	0.7799	0.7191	0.8184	0.8201	0.8776	0.7874
	$BTCP_{FAST}$	0.8091	0.8529	0.6545	0.7777	0.8412	0.8680	0.6909	0.8499	0.8569	0.9030	0.8104
	$LogTCP^{total}_{count}$	0.8061	0.8610	0.9036	0.9264	0.9408	0.9126	0.8112	0.9545	0.8961	0.9313	0.8944
	$LogTCP^{additional}_{count}$	0.8233	**0.9316**	**0.9229**	0.9298	0.9381	0.9243	0.8219	0.9255	0.9205	0.9410	0.9079
	$LogTCP^{arp}_{count}$	0.8349	0.9195	0.9206	0.9478	0.9448	0.9162	0.8202	0.9564	0.9230	0.9481	0.9132
	$LogTCP^{total}_{ordering}$	0.8046	0.8699	0.9058	0.9326	**0.9461**	0.9136	0.8105	0.9467	0.9049	0.9333	0.8968
	$LogTCP^{additional}_{ordering}$	0.8272	0.9221	0.9036	0.9292	0.9273	**0.9371**	**0.8352**	0.9548	0.9237	**0.9503**	0.9111
	$LogTCP^{arp}_{ordering}$	0.8415	0.9180	0.9183	0.9430	0.9374	0.9366	0.8208	**0.9575**	0.9186	0.9502	0.9142
	$LogTCP^{arp}_{semantics}$	**0.8612**	0.9116	0.9011	**0.9656**	0.9347	0.9274	0.8210	0.9499	**0.9537**	0.9345	**0.9161**

BTCP techniques. For example, $LogTCP^{arp}_{semantics}$ improves 17.5%, 17.4%, and 13.6% over $BTCP_{string}$, $BTCP_{topic}$, $BTCP_{FAST}$ in terms of average APFD, and improves 89.0%, 84.3%, and 52.0% over the three BTCP techniques in terms of average RAUC-25%. In particular, the worst effectiveness of our LogTCP techniques (i.e., $LogTCP^{total}_{count}$) still improves 10.0%, 39.4%, 24.2%, 15.2%, and 10.4% over

Table 4. Shapiro-Wilk normality test for each studied technique in terms of APFD

Subject	1	2	3	4	5	6	7	8	9	10
$BTCP_{string}$	✖ (0.95)	✖ (0.54)	✖ (0.83)	✖ (0.41)	✖ (0.47)	✖ (0.64)	✖ (0.69)	✔ (0.04)	✖ (0.08)	✔ (0.00)
$BTCP_{topic}$	✔ (0.00)	✖ (0.54)	✖ (0.51)	✖ (0.80)	✖ (0.20)	✖ (0.47)	✔ (0.02)	✔ (0.02)	✖ (0.78)	✔ (0.00)
$BTCP_{FAST}$	✖ (0.44)	✖ (0.91)	✖ (0.62)	✖ (0.33)	✖ (0.77)	✔ (0.02)	✖ (0.66)	✔ (0.03)	✖ (0.64)	✔ (0.00)
$WTCP_{total}$	✖ (0.78)	✖ (0.34)	✖ (0.09)	✖ (0.35)	✖ (0.54)	✖ (0.08)	✖ (0.13)	✔ (0.01)	✖ (0.80)	✔ (0.00)
$WTCP_{additional}$	✔ (0.04)	✖ (0.22)	✖ (0.15)	✖ (0.08)	✖ (0.42)	✖ (0.09)	✔ (0.01)	✔ (0.05)	✖ (0.83)	✔ (0.00)
$WTCP_{arp}$	✖ (0.08)	✖ (0.27)	✔ (0.01)	✖ (0.96)	✖ (0.70)	✖ (0.70)	✔ (0.02)	✖ (0.08)	✖ (0.25)	✖ (0.11)
$WTCP_{search}$	✖ (0.34)	✖ (0.21)	✖ (0.65)	✖ (0.33)	✖ (0.99)	✖ (0.08)	✖ (0.08)	✔ (0.03)	✖ (0.87)	✔ (0.00)
$LogTCP^{total}_{count}$	✖ (0.80)	✖ (0.73)	✖ (0.80)	✖ (0.24)	✖ (0.84)	✖ (0.12)	✖ (0.77)	✔ (0.01)	✖ (0.45)	✔ (0.00)
$LogTCP^{additional}_{count}$	✖ (0.64)	✖ (0.09)	✖ (0.56)	✖ (0.27)	✖ (0.07)	✖ (0.75)	✖ (0.23)	✔ (0.01)	✖ (0.27)	✔ (0.00)
$LogTCP^{arp}_{count}$	✖ (0.19)	✔ (0.01)	✖ (0.33)	✖ (0.85)	✖ (0.24)	✖ (0.11)	✖ (0.64)	✔ (0.00)	✖ (0.69)	✔ (0.00)
$LogTCP^{total}_{ordering}$	✖ (0.36)	✖ (0.92)	✖ (0.85)	✖ (0.22)	✖ (0.40)	✖ (0.45)	✖ (0.29)	✔ (0.00)	✖ (0.20)	✔ (0.00)
$LogTCP^{additional}_{ordering}$	✖ (0.45)	✖ (0.35)	✖ (0.42)	✖ (0.35)	✖ (0.38)	✖ (0.17)	✖ (0.42)	✔ (0.00)	✖ (0.53)	✔ (0.00)
$LogTCP^{arp}_{ordering}$	✖ (0.75)	✖ (0.06)	✖ (0.52)	✖ (0.96)	✖ (0.70)	✖ (0.12)	✖ (0.17)	✔ (0.00)	✖ (0.98)	✔ (0.00)
$LogTCP^{arp}_{semantics}$	✔ (0.03)	✖ (0.11)	✖ (0.29)	✖ (0.14)	✖ (0.86)	✔ (0.03)	✔ (0.02)	✔ (0.01)	✖ (0.56)	✔ (0.00)

the best effectiveness of the BTCP techniques (i.e., $BTCP_{FAST}$) in terms of average APFD, average RAUC-25%, average RAUC-50%, average RAUC-75%, and average RAUC-100%, respectively. The results demonstrate the significant superiority of LogTCP.

To further investigate whether our LogTCP techniques can significantly outperform the BTCP techniques in statistics, we first performed the Shapiro-Wilk normality test [101] for each studied technique (also including the WTCP techniques to be discussed in Section 5.3) on all the subjects in terms of each metric at the significance level of 0.05. Table 4 presents the p-value results in terms of APFD as the representative, since we can obtain the same conclusions from all these metrics. In this table, ✔ represents that the data conform to the normal distribution while ✖ represents that the data do not (i.e., the p-value is larger than 0.05). From this table, among 140 cases (14 techniques $\times$ 10 subjects), 73.6% (103 out of 140) do not conform to the normal distribution. Hence, we then performed the *Wilcoxon Signed-Rank Test* [115] (a popular non-parametric hypothesis test) at the significance level of 0.05 to compare each LogTCP technique with each BTCP technique in terms of each metric. Since multiple hypothesis tests may introduce p-value bias [97], we further performed the Benjamini-Hochberg method [8] to control the false discovery rate (FDR) at the FDR threshold of 0.05 in order to correct our hypothesis test results.

Here, we presented the statistical analysis results after correction in terms of APFD as the representative as shown in Table 5, since we can also obtain the same conclusions from all these metrics. In Table 5, each cell presents the p value and the statistical analysis conclusion between a pair of compared TCP techniques (i.e., the technique shown in the corresponding row and the technique shown in the corresponding column). Specifically, if the p-value is larger than 0.05, it means that the two compared techniques have no statistically significant difference in terms of APFD (marked as ❍). Otherwise, we can conclude which technique performs significantly better between them according to their APFD values in Table 3. Here, we marked ✔ for the cases where the LogTCP technique performs significantly better than the BTCP technique, and marked ✖ for the cases where the LogTCP technique performs significantly worse than the BTCP technique. From Table 5, all the cells show either ✔ or ❍, demonstrating that our LogTCP techniques *never* perform significantly worse than the BTCP techniques on all these studied subjects. In particular, among all these cases, 76.2% are marked as ✔ while 23.8% are marked as ❍, further confirming the significant superiority of our LogTCP techniques.

Table 5. Statistical analysis between LogTCP and BTCP in terms of APFD

ID	Baseline	$\text{LogTCP}^{total}_{count}$	$\text{LogTCP}^{additional}_{count}$	$\text{LogTCP}^{arp}_{count}$	$\text{LogTCP}^{total}_{ordering}$	$\text{LogTCP}^{additional}_{ordering}$	$\text{LogTCP}^{arp}_{ordering}$	$\text{LogTCP}^{arp}_{semantics}$
1	BTCP_{string}	❍ (0.37)	❍ (0.82)	❍ (0.92)	❍ (0.37)	❍ (0.90)	❍ (0.44)	❍ (0.28)
	BTCP_{topic}	❍ (0.61)	❍ (0.25)	❍ (0.22)	❍ (0.65)	❍ (0.29)	❍ (0.06)	❍ (0.07)
	BTCP_{FAST}	❍ (0.91)	❍ (0.66)	❍ (0.41)	❍ (0.96)	❍ (0.59)	❍ (0.12)	❍ (0.08)
2	BTCP_{string}	✔ (0.05)	✔ (0.00)	✔ (0.00)	✔ (0.03)	✔ (0.00)	✔ (0.00)	✔ (0.01)
	BTCP_{topic}	❍ (0.06)	✔ (0.00)	✔ (0.00)	✔ (0.04)	✔ (0.00)	✔ (0.00)	✔ (0.01)
	BTCP_{FAST}	❍ (0.92)	❍ (0.12)	❍ (0.20)	❍ (0.92)	❍ (0.16)	❍ (0.20)	❍ (0.16)
3	BTCP_{string}	❍ (0.05)	✔ (0.04)	✔ (0.04)	❍ (0.07)	✔ (0.05)	✔ (0.03)	✔ (0.01)
	BTCP_{topic}	✔ (0.01)	✔ (0.02)	✔ (0.01)	✔ (0.01)	✔ (0.01)	✔ (0.01)	✔ (0.03)
	BTCP_{FAST}	✔ (0.01)	✔ (0.01)	✔ (0.01)	✔ (0.01)	✔ (0.01)	✔ (0.01)	❍ (0.06)
4	BTCP_{string}	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)
	BTCP_{topic}	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)
	BTCP_{FAST}	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)
5	BTCP_{string}	✔ (0.02)	✔ (0.03)	✔ (0.02)	✔ (0.01)	✔ (0.01)	✔ (0.02)	✔ (0.04)
	BTCP_{topic}	✔ (0.01)	✔ (0.03)	✔ (0.01)	✔ (0.01)	✔ (0.01)	✔ (0.01)	✔ (0.04)
	BTCP_{FAST}	✔ (0.03)	❍ (0.05)	✔ (0.04)	✔ (0.02)	❍ (0.07)	✔ (0.04)	❍ (0.19)
6	BTCP_{string}	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)
	BTCP_{topic}	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)
	BTCP_{FAST}	❍ (0.27)	❍ (0.20)	✔ (0.05)	❍ (0.30)	❍ (0.07)	✔ (0.04)	✔ (0.01)
7	BTCP_{string}	❍ (0.07)	✔ (0.02)	✔ (0.03)	❍ (0.08)	✔ (0.01)	✔ (0.04)	✔ (0.01)
	BTCP_{topic}	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)
	BTCP_{FAST}	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)
8	BTCP_{string}	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)
	BTCP_{topic}	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)
	BTCP_{FAST}	✔ (0.00)	✔ (0.01)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)
9	BTCP_{string}	✔ (0.02)	✔ (0.01)	✔ (0.01)	✔ (0.01)	✔ (0.01)	✔ (0.01)	✔ (0.01)
	BTCP_{topic}	❍ (0.11)	✔ (0.03)	✔ (0.01)	❍ (0.06)	✔ (0.02)	✔ (0.03)	✔ (0.03)
	BTCP_{FAST}	❍ (0.30)	❍ (0.27)	❍ (0.20)	❍ (0.24)	❍ (0.17)	❍ (0.20)	❍ (0.05)
10	BTCP_{string}	✔ (0.03)	✔ (0.00)	✔ (0.00)	✔ (0.02)	✔ (0.00)	✔ (0.00)	✔ (0.00)
	BTCP_{topic}	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)
	BTCP_{FAST}	✔ (0.02)	✔ (0.00)	✔ (0.00)	✔ (0.01)	✔ (0.00)	✔ (0.00)	✔ (0.01)

Finding 1: All of our LogTCP techniques perform better than all the studied BTCP techniques in terms of both average APFD and average RAUC-s, and the vast majority of outperformance cases are statistically significant, demonstrating that LogTCP is indeed able to largely improve the effectiveness of BTCP.

5.2 RQ2: Influence of Inherent Factors in LogTCP

In LogTCP, there are two inherent factors, i.e., log representation strategies and test case prioritization strategies, and here we investigated their influence respectively in order to recommend how to apply and further improve these LogTCP techniques in practice.

To investigate the influence of log representation strategies, we divided our LogTCP techniques into three groups for comparison by controlling another inherent factor. That is, the techniques in the same comparison group have the same test case prioritization strategy, which aims to get rid of the influence of test case prioritization strategies. More specifically, we put $\text{LogTCP}^{total}_{count}$, $\text{LogTCP}^{total}_{ordering}$ into a comparison group, $\text{LogTCP}^{additional}_{count}$, $\text{LogTCP}^{additional}_{ordering}$ into a comparison group, and $\text{LogTCP}^{arp}_{semantics}$, $\text{LogTCP}^{arp}_{count}$, $\text{LogTCP}^{arp}_{ordering}$ into a comparison group. From Table 3, we still found that $\text{LogTCP}^{total}_{ordering}$ performs better than $\text{LogTCP}^{total}_{count}$ and $\text{LogTCP}^{additional}_{ordering}$ performs better than $\text{LogTCP}^{additional}_{count}$ in terms of all the metrics on average, although the differences are relatively small. Among $\text{LogTCP}^{arp}_{semantics}$, $\text{LogTCP}^{arp}_{count}$, and $\text{LogTCP}^{arp}_{ordering}$, $\text{LogTCP}^{arp}_{semantics}$ performs the best

in terms of all the metrics on average while LogTCP$^{arp}_{count}$ performs the worst (except in RAUC-75%). That is, we can conclude that with the same test case prioritization strategy, the semantics-based log representation strategy performs the best, and the ordering-based log representation strategy outperforms the count-based log representation strategy. The results indicate that the ordering and semantics of log events indeed contribute to the effectiveness of LogTCP compared with the count features of log events used in count-based log representation. It is reasonable since the test behaviors of a test case tend to involve a series of *continuous actions* (that can be captured by both ordering and semantics of log events), rather than simply treat each action independently. In the future, it is promising to design more effective strategies to represent the ordering and semantics of log events, in order to further improve the effectiveness of LogTCP.

Similarly, to investigate the influence of test case prioritization strategies, we divided our LogTCP techniques into two groups and the techniques in the same group have the same log representation strategy. Here, we did not study LogTCP$^{arp}_{semantics}$ since only this technique uses the semantics-based log representation strategy. That is, we put LogTCP$^{total}_{count}$, LogTCP$^{additional}_{count}$, LogTCP$^{arp}_{count}$ into a comparison group, and LogTCP$^{total}_{ordering}$, LogTCP$^{additional}_{ordering}$, LogTCP$^{arp}_{ordering}$ into a comparison group. From Table 3, on average, LogTCP$^{arp}_{count}$ performs the best while LogTCP$^{total}_{count}$ performs the worst in terms of all the metrics among the three techniques with the count-based log representation strategy. Same as the group of techniques with the ordering-based log representation strategy, on average, LogTCP$^{arp}_{ordering}$ performs the best while LogTCP$^{total}_{ordering}$ performs the worst in terms of all the metrics. That is, regardless of on the basis of count-based log representation or ordering-based log representation, the ARP strategy is the most effective while the total prioritization strategy is the least effective. In particular, the effectiveness differences between the total-based techniques and the additional-based techniques are more obvious than those between the additional-based techniques and the ARP-based techniques. The reason could be that both the ARP strategy and the additional strategy consider the diversity among test cases in prioritization, while the total strategy considers each test case independently which could cause the test cases prioritized closely have large overlaps in fault detection and thus damage the overall effectiveness.

To sum up, the semantics and ordering log representation strategies are more effective in the three comparison groups with different test case prioritization strategies, and the ARP strategy is more effective in the two comparison groups with different log representation strategies. By combining these more effective log representation strategies with the more effective test case prioritization strategy respectively, we obtain LogTCP$^{arp}_{ordering}$ and LogTCP$^{arp}_{semantics}$. Indeed, both of them perform better than the other five LogTCP techniques in terms of all these metrics on average (except LogTCP$^{arp}_{ordering}$ in terms of RAUC-75%, which is slightly worse than LogTCP$^{arp}_{count}$). Therefore, when applying LogTCP to the practice, either LogTCP$^{arp}_{ordering}$ or LogTCP$^{arp}_{semantics}$ can be the first choice. In particular, on average, LogTCP$^{arp}_{semantics}$ performs slightly better than LogTCP$^{arp}_{ordering}$ in terms of all the metrics.

> **Finding 2**: In terms of these metrics on average, semantics-based log representation and ordering-based log representation are more effective than count-based log representation, and the ARP test case prioritization strategy is more effective than the total and additional strategies. Thus, we recommend LogTCP$^{arp}_{semantics}$ and LogTCP$^{arp}_{ordering}$ as the representatives of LogTCP in practice.

In addition, due to the effectiveness of the ARP test case prioritization strategy, we also investigated the influence of different distances on its effectiveness. Here, we studied three popular

Table 6. Effectiveness of LogTCP with different distances in the ARP strategy in terms of APFD

Project	$\text{LogTCP}^{arp}_{count}$			$\text{LogTCP}^{arp}_{ordering}$			$\text{LogTCP}^{arp}_{semantics}$		
	Euclidean	Manhattan	Cosine	Euclidean	Manhattan	Cosine	Euclidean	Manhattan	Cosine
1	0.7503	**0.7576**	0.6617	**0.7557**	0.7423	0.6985	**0.7740**	0.7661	0.7086
2	**0.7478**	0.7412	0.7103	0.7466	**0.7481**	0.6879	**0.7496**	0.7198	0.7183
3	**0.7284**	0.7267	0.7187	**0.7267**	0.7258	0.7062	**0.7444**	0.7142	0.6964
4	**0.8476**	**0.8476**	0.8136	**0.8433**	0.8421	0.8173	**0.8633**	**0.8633**	0.8300
5	**0.7005**	**0.7005**	0.6801	**0.6950**	0.6891	0.6702	**0.6931**	0.6851	0.6772
6	0.7852	**0.7892**	0.7853	**0.8084**	0.7985	0.7858	**0.8187**	0.7989	0.8009
7	0.7129	**0.7136**	0.7122	**0.7135**	0.7078	0.7121	**0.7136**	**0.7136**	0.6935
8	0.8823	**0.8866**	0.8510	0.8834	**0.8906**	0.8363	0.8764	**0.8791**	0.8758
9	**0.7878**	**0.7878**	0.7862	**0.7841**	0.7825	0.7635	**0.8132**	0.7720	0.7222
10	0.9364	**0.9401**	0.9248	0.9385	**0.9386**	0.9079	**0.9230**	0.9194	0.9131
Average	0.7879	**0.7891**	0.7644	**0.7895**	0.7865	0.7586	**0.7969**	0.7832	0.7636

distances and used the APFD metric as the representative, whose results are shown in Table 6. In this table, we marked the best distance in each technique on each subject as **bold**. From Table 6, in all three techniques, Cosine distance performs the worst on average. In $\text{LogTCP}^{arp}_{ordering}$ and $\text{LogTCP}^{arp}_{semantics}$, Euclidean distance performs better than Manhattan distance on average, while in $\text{LogTCP}^{arp}_{count}$, Manhattan distance is more effective on average. Since the superiority of $\text{LogTCP}^{arp}_{ordering}$ and $\text{LogTCP}^{arp}_{semantics}$ among all the LogTCP techniques, Euclidean distance is the first choice in the ARP test case prioritization strategy in practice.

Finding 3: In our recommended $\text{LogTCP}^{arp}_{semantics}$ and $\text{LogTCP}^{arp}_{ordering}$ techniques, Euclidean distance makes the ARP test case prioritization strategy more effective than both Manhattan distance and Cosine distance, and thus is recommended as the default distance in the ARP strategy in practice.

5.3 RQ3: LogTCP v.s. Existing WTCP Techniques

We have confirmed that LogTCP can effectively improve the effectiveness of BTCP above. In RQ3, we further investigated the effectiveness gap to WTCP. To answer this RQ, we compared LogTCP with four representative WTCP techniques. Table 7 shows the comparison results in terms of average APFD and average RAUC-s across all the faulty versions for each subject. We marked the best result as **bold** for each subject in terms of each metric in this table.

From Table 7, surprisingly, our log-based techniques achieve the best results on nearly half of the subjects compared with the representative WTCP techniques in terms of each metric. For example, in terms of APFD, LogTCP achieves the best results on five subjects while WTCP techniques achieve the best results on five subjects. In particular, $\text{LogTCP}^{arp}_{semantics}$ performs the best among all the log-based and studied WTCP techniques in terms of average APFD, average RAUC-25%, and average RAUC-50%. Also, in terms of average RAUC-75% and average RAUC-100%, $\text{LogTCP}^{arp}_{semantics}$ is just slightly less effective than the best WTCP technique (i.e., $\text{WTCP}_{additional}$). The results demonstrate that LogTCP is indeed effective to bridge the effectiveness gap between BTCP and WTCP, even slightly outperforming the state-of-the-art WTCP techniques on many subjects.

To further investigate whether there are statistically significant differences between LogTCP and WTCP, we also performed the Wilcoxon Signed-Rank Test at the significance level of 0.05

Table 7. Comparison between LogTCP and existing WTCP techniques

Metrics	Approach	1	2	3	4	5	6	7	8	9	10	Average
APFD	$WTCP_{total}$	0.6973	0.7576	0.6893	0.8621	0.6866	0.8041	0.7363	**0.8981**	0.7471	0.9109	0.7789
	$WTCP_{additional}$	0.7642	0.7722	0.6813	0.8530	0.6995	0.8225	**0.7808**	0.8792	0.7333	**0.9478**	0.7934
	$WTCP_{arp}$	0.7082	0.6740	0.5818	0.7027	0.5864	0.7184	0.6671	0.6948	0.7180	0.8264	0.6878
	$WTCP_{search}$	0.7274	**0.7844**	0.6867	0.8530	0.6970	**0.8245**	0.7699	0.8815	0.7116	0.9102	0.7846
	$LogTCP^{total}_{count}$	0.7208	0.7004	0.7151	0.8282	0.6965	0.7821	0.7052	0.8806	0.7651	0.9198	0.7714
	$LogTCP^{additional}_{count}$	0.7362	0.7576	0.7302	0.8312	**0.7005**	0.7921	0.7145	0.8540	0.7857	0.9293	0.7831
	$LogTCP^{arp}_{count}$	0.7503	0.7478	0.7284	0.8476	**0.7005**	0.7852	0.7129	0.8823	0.7878	0.9364	0.7879
	$LogTCP^{total}_{ordering}$	0.7194	0.7076	0.7169	0.8336	0.6990	0.7886	0.7045	0.8734	0.7725	0.9218	0.7737
	$LogTCP^{additional}_{ordering}$	0.7397	0.7499	0.7151	0.8306	0.6896	0.8030	0.7260	0.8809	0.7884	0.9385	0.7862
	$LogTCP^{arp}_{ordering}$	0.7557	0.7466	0.7267	0.8433	0.6950	0.8084	0.7135	0.8834	0.7841	0.9385	0.7895
	$LogTCP^{arp}_{semantics}$	**0.7740**	0.7496	**0.7444**	**0.8633**	0.6931	0.8187	0.7136	0.8764	**0.8132**	0.9230	**0.7969**
RAUC-25%	$WTCP_{total}$	0.5044	0.7953	0.6410	**0.9913**	0.7900	0.8299	0.7806	**0.9598**	0.7845	0.8276	0.7904
	$WTCP_{additional}$	0.6833	0.8510	0.5353	0.8783	0.7854	**0.8815**	**0.8725**	0.9174	0.6979	**0.8937**	0.7996
	$WTCP_{arp}$	0.4777	0.5949	0.2917	0.3938	0.4909	0.5578	0.4879	0.3206	0.6219	0.5103	0.4748
	$WTCP_{search}$	0.5932	**0.8703**	0.5353	0.8767	0.7785	0.8759	0.7880	0.9342	0.6979	0.7972	0.7747
	$LogTCP^{total}_{count}$	0.5205	0.5113	0.8814	0.8866	**0.9361**	0.7534	0.6111	0.9358	0.7208	0.8471	0.7604
	$LogTCP^{additional}_{count}$	0.6316	0.7438	**0.9583**	0.9005	**0.9361**	0.7835	0.6882	0.8626	0.7880	0.8556	0.8148
	$LogTCP^{arp}_{count}$	0.6215	0.7395	0.9199	0.9252	**0.9361**	0.7803	0.7421	0.9268	0.7898	0.8578	0.8239
	$LogTCP^{total}_{ordering}$	0.5526	0.5241	0.9006	0.9127	0.9292	0.7751	0.6127	0.9291	0.7208	0.8505	0.7707
	$LogTCP^{additional}_{ordering}$	0.6807	0.7224	0.9231	0.8988	0.9064	0.8292	0.7043	0.9280	0.7650	0.8671	0.8225
	$LogTCP^{arp}_{ordering}$	**0.7094**	0.7353	0.9135	0.9200	0.9064	0.8201	0.7116	0.9327	0.7580	0.8670	0.8274
	$LogTCP^{arp}_{semantics}$	0.6919	0.7402	0.8944	0.9704	0.8493	0.8257	0.6789	0.9073	**0.9149**	0.8184	**0.8291**
RAUC-50%	$WTCP_{total}$	0.6108	0.8386	0.7582	0.9630	0.8672	0.8736	0.8004	**0.9666**	0.8095	0.8705	0.8358
	$WTCP_{additional}$	0.7616	0.8781	0.7227	0.9275	0.8832	0.9098	**0.8617**	0.9344	0.7727	**0.9284**	0.8580
	$WTCP_{arp}$	0.6285	0.6668	0.5191	0.5807	0.5849	0.6909	0.5928	0.5291	0.7443	0.6848	0.6222
	$WTCP_{search}$	0.6953	**0.9102**	0.7336	0.8998	0.8782	**0.9192**	0.8316	0.9385	0.7663	0.8590	0.8432
	$LogTCP^{total}_{count}$	0.6646	0.7033	0.8989	0.8927	0.9126	0.8302	0.7071	0.9436	0.8194	0.8836	0.8256
	$LogTCP^{additional}_{count}$	0.7094	0.8495	**0.9303**	0.8934	**0.9160**	0.8525	0.7312	0.8798	0.8775	0.8981	0.8538
	$LogTCP^{arp}_{count}$	0.7221	0.8416	0.9180	0.9243	0.9143	0.8424	0.7536	0.9497	0.8732	0.9069	0.8646
	$LogTCP^{total}_{ordering}$	0.6591	0.7258	0.8989	0.9053	0.9092	0.8367	0.7052	0.9405	0.8279	0.8872	0.8296
	$LogTCP^{additional}_{ordering}$	0.7132	0.8377	0.8948	0.8998	0.8958	0.8760	0.7457	0.9501	0.8661	0.9099	0.8589
	$LogTCP^{arp}_{ordering}$	0.7482	0.8430	0.9180	0.9180	0.8958	0.8701	0.7419	0.9528	0.8633	0.9109	0.8662
	$LogTCP^{arp}_{semantics}$	**0.7813**	0.8184	0.8828	**0.9716**	0.8891	0.8726	0.7342	0.9382	**0.9447**	0.8818	**0.8715**
RAUC-75%	$WTCP_{total}$	0.6991	0.8981	0.8333	0.9588	0.9065	0.9081	0.8093	**0.9687**	0.8493	0.9001	0.8731
	$WTCP_{additional}$	0.8025	0.9239	0.8178	0.9449	0.9276	0.9375	**0.8709**	0.9396	0.8272	**0.9479**	**0.8940**
	$WTCP_{arp}$	0.7205	0.7590	0.6413	0.7198	0.7124	0.7848	0.6895	0.6718	0.8060	0.7840	0.7289
	$WTCP_{search}$	0.7569	**0.9454**	0.8260	0.9281	0.9247	**0.9474**	0.8528	0.9432	0.7967	0.8986	0.8820
	$LogTCP^{total}_{count}$	0.7467	0.8044	0.8897	0.9087	0.9243	0.8838	0.7557	0.9441	0.8710	0.9102	0.8639
	$LogTCP^{additional}_{count}$	0.7699	0.9023	**0.9167**	0.9129	0.9247	0.9011	0.7687	0.9045	0.9053	0.9226	0.8829
	$LogTCP^{arp}_{count}$	0.7858	0.8876	0.9085	0.9352	**0.9372**	0.8910	0.7764	0.9498	0.9032	0.9327	0.8907
	$LogTCP^{total}_{ordering}$	0.7435	0.8170	0.8930	0.9171	0.9291	0.8877	0.7557	0.9399	0.8773	0.9130	0.8673
	$LogTCP^{additional}_{ordering}$	0.7754	0.8887	0.8881	0.9124	0.9123	0.9176	0.7875	0.9479	0.9003	0.9350	0.8865
	$LogTCP^{arp}_{ordering}$	0.7935	0.8882	0.9085	0.9296	0.9180	0.9145	0.7699	0.9494	0.8986	0.9354	0.8906
	$LogTCP^{arp}_{semantics}$	**0.8216**	0.8739	0.8808	**0.9593**	0.9195	0.9093	0.7708	0.9393	**0.9469**	0.9146	0.8936
RAUC-100%	$WTCP_{total}$	0.7758	0.9287	0.8758	0.9643	0.9283	0.9321	0.8470	**0.9735**	0.8786	0.9223	0.9026
	$WTCP_{additional}$	0.8503	0.9467	0.8656	0.9540	0.9458	0.9534	**0.8982**	0.9530	0.8623	**0.9597**	**0.9189**
	$WTCP_{arp}$	0.7915	0.8276	0.7413	0.7873	0.7927	0.8390	0.7676	0.7554	0.8457	0.8384	0.7987
	$WTCP_{search}$	0.8117	**0.9618**	0.8724	0.9455	0.9424	**0.9596**	0.8858	0.9550	0.8367	0.9235	0.9094
	$LogTCP^{total}_{count}$	0.8061	0.8610	0.9036	0.9264	0.9408	0.9126	0.8112	0.9545	0.8961	0.9313	0.8944
	$LogTCP^{additional}_{count}$	0.8233	0.9316	**0.9229**	0.9298	0.9381	0.9243	0.8219	0.9255	0.9205	0.9410	0.9079
	$LogTCP^{arp}_{count}$	0.8349	0.9195	0.9206	0.9478	0.9448	0.9162	0.8202	0.9564	0.9230	0.9481	0.9132
	$LogTCP^{total}_{ordering}$	0.8046	0.8699	0.9058	0.9326	**0.9461**	0.9136	0.8105	0.9467	0.9049	0.9333	0.8968
	$LogTCP^{additional}_{ordering}$	0.8272	0.9221	0.9036	0.9292	0.9273	0.9371	0.8352	0.9548	0.9237	0.9503	0.9111
	$LogTCP^{arp}_{ordering}$	0.8415	0.9180	0.9183	0.9430	0.9374	0.9366	0.8208	0.9575	0.9186	0.9502	0.9142
	$LogTCP^{arp}_{semantics}$	**0.8612**	0.9116	0.9011	**0.9656**	0.9347	0.9274	0.8210	0.9499	**0.9537**	0.9345	0.9161

Table 8. Statistical analysis between LogTCP and WTCP in terms of APFD

ID	Baseline	$LogTCP^{total}_{count}$	$LogTCP^{additional}_{count}$	$LogTCP^{arp}_{count}$	$LogTCP^{total}_{ordering}$	$LogTCP^{additional}_{ordering}$	$LogTCP^{arp}_{ordering}$	$LogTCP^{arp}_{semantics}$
1	$WTCP_{total}$	❍ (0.54)	❍ (0.40)	❍ (0.26)	❍ (0.58)	❍ (0.34)	❍ (0.08)	❍ (0.06)
	$WTCP_{additional}$	❍ (0.35)	❍ (0.62)	❍ (0.65)	❍ (0.33)	❍ (0.69)	❍ (0.85)	❍ (0.91)
	$WTCP_{arp}$	❍ (0.73)	❍ (0.56)	❍ (0.25)	❍ (0.74)	❍ (0.54)	❍ (0.23)	✔ (0.04)
	$WTCP_{search}$	❍ (0.65)	❍ (0.93)	❍ (0.62)	❍ (0.68)	❍ (1.00)	❍ (0.56)	❍ (0.27)
2	$WTCP_{total}$	❍ (0.18)	❍ (0.93)	❍ (0.82)	❍ (0.20)	❍ (0.61)	❍ (0.72)	❍ (0.78)
	$WTCP_{additional}$	✖ (0.01)	❍ (0.56)	❍ (0.41)	✖ (0.02)	❍ (0.36)	❍ (0.47)	❍ (0.47)
	$WTCP_{arp}$	❍ (0.57)	❍ (0.06)	❍ (0.07)	❍ (0.38)	❍ (0.06)	❍ (0.12)	❍ (0.06)
	$WTCP_{search}$	✖ (0.01)	❍ (0.32)	❍ (0.18)	✖ (0.01)	❍ (0.22)	❍ (0.18)	❍ (0.32)
3	$WTCP_{total}$	❍ (0.65)	❍ (0.59)	❍ (0.72)	❍ (0.56)	❍ (0.90)	❍ (0.61)	❍ (0.28)
	$WTCP_{additional}$	❍ (0.61)	❍ (0.51)	❍ (0.65)	❍ (0.56)	❍ (0.65)	❍ (0.47)	❍ (0.24)
	$WTCP_{arp}$	✔ (0.05)	✔ (0.05)	✔ (0.03)	✔ (0.03)	✔ (0.03)	✔ (0.02)	✔ (0.02)
	$WTCP_{search}$	❍ (0.78)	❍ (0.65)	❍ (0.72)	❍ (0.65)	❍ (0.78)	❍ (0.59)	❍ (0.29)
4	$WTCP_{total}$	❍ (0.54)	❍ (0.63)	❍ (0.51)	❍ (0.56)	❍ (0.58)	❍ (0.54)	❍ (1.00)
	$WTCP_{additional}$	❍ (0.78)	❍ (0.79)	❍ (0.79)	❍ (0.90)	❍ (0.78)	❍ (0.75)	❍ (0.66)
	$WTCP_{arp}$	✔ (0.01)	✔ (0.02)	✔ (0.01)	✔ (0.01)	✔ (0.01)	✔ (0.01)	✔ (0.01)
	$WTCP_{search}$	❍ (0.70)	❍ (0.90)	❍ (0.93)	❍ (0.79)	❍ (0.77)	❍ (0.90)	❍ (0.57)
5	$WTCP_{total}$	❍ (0.65)	❍ (0.60)	❍ (0.69)	❍ (0.63)	❍ (1.00)	❍ (0.90)	❍ (0.93)
	$WTCP_{additional}$	❍ (0.63)	❍ (0.48)	❍ (0.91)	❍ (0.63)	❍ (0.73)	❍ (0.93)	❍ (1.00)
	$WTCP_{arp}$	❍ (0.23)	❍ (0.23)	❍ (0.18)	❍ (0.20)	❍ (0.17)	❍ (0.21)	❍ (0.18)
	$WTCP_{search}$	❍ (0.93)	❍ (0.98)	❍ (0.99)	❍ (0.99)	❍ (0.91)	❍ (1.00)	❍ (0.91)
6	$WTCP_{total}$	❍ (0.56)	❍ (0.57)	❍ (1.00)	❍ (0.56)	❍ (0.91)	❍ (0.73)	❍ (0.84)
	$WTCP_{additional}$	❍ (0.32)	❍ (0.36)	❍ (0.32)	❍ (0.34)	❍ (0.56)	❍ (0.51)	❍ (0.93)
	$WTCP_{arp}$	✔ (0.05)	✔ (0.02)	✔ (0.04)	✔ (0.05)	✔ (0.01)	✔ (0.02)	✔ (0.00)
	$WTCP_{search}$	❍ (0.41)	❍ (0.36)	❍ (0.51)	❍ (0.36)	❍ (0.60)	❍ (0.65)	❍ (0.96)
7	$WTCP_{total}$	✖ (0.02)	❍ (0.27)	❍ (0.18)	✖ (0.04)	❍ (0.54)	❍ (0.18)	❍ (0.13)
	$WTCP_{additional}$	✖ (0.00)	✖ (0.00)	✖ (0.00)	✖ (0.00)	✖ (0.00)	✖ (0.00)	✖ (0.00)
	$WTCP_{arp}$	❍ (0.29)	❍ (0.09)	❍ (0.09)	❍ (0.18)	✔ (0.01)	❍ (0.10)	❍ (0.08)
	$WTCP_{search}$	✖ (0.00)	✖ (0.01)	✖ (0.00)	✖ (0.00)	✖ (0.02)	✖ (0.00)	✖ (0.01)
8	$WTCP_{total}$	✖ (0.05)	✖ (0.00)	❍ (0.30)	✖ (0.01)	❍ (0.17)	❍ (0.18)	✖ (0.02)
	$WTCP_{additional}$	❍ (0.93)	❍ (0.24)	❍ (0.82)	❍ (0.75)	❍ (0.90)	❍ (0.79)	❍ (0.93)
	$WTCP_{arp}$	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)
	$WTCP_{search}$	❍ (0.76)	❍ (0.11)	❍ (0.99)	❍ (0.54)	❍ (0.93)	❍ (0.90)	❍ (0.90)
9	$WTCP_{total}$	❍ (0.87)	❍ (0.69)	❍ (0.64)	❍ (0.69)	❍ (0.61)	❍ (0.73)	❍ (0.32)
	$WTCP_{additional}$	❍ (0.56)	❍ (0.54)	❍ (0.47)	❍ (0.56)	❍ (0.44)	❍ (0.47)	❍ (0.30)
	$WTCP_{arp}$	❍ (0.38)	❍ (0.14)	❍ (0.11)	❍ (0.30)	❍ (0.12)	❍ (0.17)	✔ (0.05)
	$WTCP_{search}$	❍ (0.51)	❍ (0.44)	❍ (0.38)	❍ (0.47)	❍ (0.41)	❍ (0.41)	❍ (0.17)
10	$WTCP_{total}$	✔ (0.00)	❍ (0.09)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	❍ (0.41)
	$WTCP_{additional}$	✖ (0.00)	✖ (0.00)	✖ (0.01)	✖ (0.01)	❍ (0.09)	✖ (0.02)	✖ (0.00)
	$WTCP_{arp}$	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)	✔ (0.00)
	$WTCP_{search}$	❍ (0.54)	❍ (0.16)	✔ (0.04)	❍ (0.45)	✔ (0.03)	✔ (0.02)	❍ (0.34)

Table 9. Spearman-rank correlation between the APFD result and the subject size

Technique	coefficient	p-value
$LogTCP^{total}_{count}$	0.20	0.58
$LogTCP^{additional}_{count}$	0.30	0.40
$LogTCP^{arp}_{count}$	0.18	0.63
$LogTCP^{total}_{ordering}$	0.18	0.63
$LogTCP^{additional}_{ordering}$	0.39	0.26
$LogTCP^{arp}_{ordering}$	0.25	0.49
$LogTCP^{arp}_{semantics}$	0.25	0.49

to compare each LogTCP technique with each WTCP technique, since most of our data do not conform to the normal distribution according to Table 4. To correct the multiple hypothesis test results, we also performed the Benjamini-Hochberg method [8] to control the FDR. We also used

Table 10. Influence of logging levels on the effectiveness of $\text{LogTCP}^{arp}_{semantics}$ in terms of APFD

Technique	Level	1	2	3	4	5	6	7	8	9	10	Average
$\text{LogTCP}^{arp}_{semantics}$	ALL	0.7740	0.7496	0.7444	0.8633	0.6931	0.8187	0.7136	0.8764	0.8132	0.9230	0.7969
	DEBUG	0.7564	0.7801	0.7142	0.8542	0.6866	0.8101	0.7061	0.8734	0.7677	0.9149	0.7864
	INFO	0.7192	0.7734	0.7373	0.8512	0.6722	0.8010	0.7155	0.8372	0.7979	0.9098	0.7815
	WARN	0.7296	0.7651	0.7053	0.8555	0.6568	0.7890	0.6924	0.8151	0.7910	0.9000	0.7700
	ERROR	0.6999	0.6744	0.7311	0.8445	0.6931	0.7343	0.6695	0.7254	—	0.8983	0.7412
BTCP_{string}		0.7442	0.6561	0.5373	0.6361	0.5630	0.6031	0.6616	0.7980	0.6778	0.9060	0.6783
BTCP_{topic}		0.6966	0.6772	0.5649	0.6676	0.5764	0.6707	0.6250	0.7524	0.6968	0.8620	0.6790
BTCP_{FAST}		0.7223	0.7016	0.5364	0.6888	0.6213	0.7408	0.6005	0.7809	0.7296	0.8915	0.7014

the APFD metric as the representative (we can obtain the same conclusions from all these metrics) and reported the statistical analysis results after correction in Table 8. Here, we marked ✔ for the cases where the LogTCP technique performs significantly better than the WTCP technique, ✖ for the cases where the LogTCP technique performs significantly worse than the WTCP technique, and ❍ for the cases where they have no significant difference in statistics. From Table 8, among all the cases, only 10.7% are marked as ✖ and even 16.4% are marked as ✔, indicating that LogTCP significantly bridges the effectiveness gap between BTCP and WTCP. The results demonstrate that we are indeed able to have the best of both BTCP and WTCP (i.e., achieving great effectiveness without relying on source code information) through log analysis.

From Table 8, we found that LogTCP performs significantly worse than WTCP on some subjects (i.e., subjects 7 and 10). Actually, LogTCP belongs to BTCP and thus this phenomenon is as expected, but it is also important to understand the reason behind this phenomenon in order to further improve LogTCP in the future. First of all, we found that the two subjects are more large-scale than other subjects, and thus we suspect whether there is a correlation between the effectiveness of LogTCP and the size of the subject under test. Specifically, we measured the Spearman-rank correlation [105] between the APFD result and the subject size for each log-based TCP technique, whose results are shown in Table 9. We found that all the p-values are larger than 0.05 and all the coefficients are smaller than 0.39, indicating that their is no statistically significant correlation between them. Then, we conducted manual analysis on the logs of these subjects for further investigation. We found that for the two subjects, the log events extracted by Drain3 (the used log parser in our study) are not very accurate, i.e., some log parameters are not accurately removed from log messages. In other words, some log messages should belong to the same log event, but are processed into different log events due to the inaccuracy of Drain3. Such inaccuracy can have negative influence on the effectiveness of LogTCP, which may be the main reason why LogTCP performs worse than WTCP on the two subjects. In the future, we will incorporate more advanced log parsing methods into LogTCP to further improve the effectiveness of LogTCP by improving the accuracy of log parsing.

Finding 4: Indeed, LogTCP is able to significantly bridge the effectiveness gap between BTCP and WTCP. Even, $\text{LogTCP}^{arp}_{semantics}$ performs better than the state-of-the-art WTCP technique in terms of average APFD, average RAUC-25%, and average RAUC-50%. Thus, LogTCP could be more practical since it can achieve competitive effectiveness to the state-of-the-art WTCP without relying on source code information.

6 DISCUSSION

6.1 Influence of Logging Levels

In practice, there are several logging levels that can indicate how important a log message is. For example, the logging levels provided by the used logging framework (i.e., Log4j or Logback) in our study include ALL, DEBUG, INFO, WARN, ERROR, etc. These logging levels have the partial ordering relation: ALL < DEBUG < INFO < WARN < ERROR < . . .. When setting the level to a lower one (e.g., DEBUG), all the log messages with the same and higher levels (e.g., DEBUG, INFO, WARN, ERROR, etc) can be produced. In our study, we set the logging level to ALL (i.e., the lowest one) for all the subjects, which can produce log messages with various levels. In this way, the dynamic behaviors of test cases can be reflected more sufficiently.

To better understand why logs can facilitate the task of test case prioritization, we conducted an experiment to investigate the influence of different logging levels on the effectiveness of LogTCP. Besides our used logging level (i.e., ALL) in the study, we further studied the effectiveness of LogTCP when setting the logging level to DEBUG, INFO, WARN, and ERROR, respectively. These logging levels are commonly-used in practice as presented in the existing work [5], where ALL can record the most log messages while ERROR can record the fewest among the five studied levels. In this experiment, we took $\text{LogTCP}^{arp}_{semantics}$ as the representative due to its effectiveness as demonstrated in Section 5.2. Table 10 shows the APFD results of $\text{LogTCP}^{arp}_{semantics}$ under the settings of different logging levels, where "—" indicates that there is no log message produced under the setting of the corresponding logging level. From this table, we found that $\text{LogTCP}^{arp}_{semantics}$ performs better in terms of APFD when a lower logging level is set, e.g., the average APFD results are 0.7969, 0.7864, 0.7815, 0.7700, and 0.7412 when setting the logging level to ALL, DEBUG, INFO, WARN, ERROR, respectively. One major reason is that more adequate log messages can be produced under the setting of lower logging levels, which can better reflect the dynamic behaviors of test cases. In particular, no matter which logging level is set among the five, $\text{LogTCP}^{arp}_{semantics}$ always outperforms the existing BTCP techniques in terms of APFD, demonstrating the power of logs for test case prioritization.

Overall, one major reason why LogTCP can improve the effectiveness of BTCP is that logs can effectively reflect the dynamic behaviors of test cases, which can provide more accurate information to distinguish the difference between test cases. More adequate log messages can better reflect the dynamic behaviors of test cases, and thus can make more contributions to the effectiveness of test case prioritization. During the practical use of LogTCP, we recommend to set the logging level to the lowest one (e.g., ALL in Log4j or Logback). If the testing resource is limited, setting higher logging levels can also achieve better TCP effectiveness than the existing BTCP techniques.

6.2 Exploring Log-based Failure Detection for Test Case Prioritization

In the literature, there are many log-based failure detection techniques, such as DeepLog [32], LogBert[39] and PLELog[117], which aim to detect failures in system runtime by building a machine learning or deep learning model based on a large amount of log data. Intuitively, they can be also adapted to the task of test case prioritization by using the model to predict how likely a test case reveals failures based on its corresponding logs and then prioritizing test cases based on the prediction results. Hence, it is interesting to investigate whether directly adapting the existing log-based failure detection techniques can also perform well for the task of test case prioritization.

To obtain sufficient training data and avoid data leakage, we adopted EvoSuite [36], a state-of-the-art test case generation tool, to generate a large number of test cases for each studied subject, and collected the corresponding log data by running them. Since almost all the test cases generated by EvoSuite are passing test cases, we used all the corresponding normal logs as training data to

Table 11. Effectiveness of the existing log-based failure detection techniques for TCP in terms of APFD

Technique	Blueflood	Flume	Wicket	Average
DeepLog	0.6884	0.6801	0.9113	0.7599
PCA	0.7124	0.5963	0.9141	0.7409
LogCluster	0.6680	0.6926	0.8825	0.7477
LogTCP$^{total}_{count}$	0.7151	0.6965	0.9198	0.7771
LogTCP$^{additional}_{count}$	0.7302	**0.7005**	0.9293	0.7867
LogTCP$^{arp}_{count}$	0.7284	**0.7005**	0.9364	**0.7884**
LogTCP$^{total}_{ordering}$	0.7169	0.6990	0.9218	0.7792
LogTCP$^{additional}_{ordering}$	0.7151	0.6896	**0.9385**	0.7811
LogTCP$^{arp}_{ordering}$	0.7267	0.6950	**0.9385**	0.7867
LogTCP$^{arp}_{semantics}$	**0.7444**	0.6931	0.9230	0.7868

build the failure detection model for each studied subject. That is, we have to adopt semi-supervised or unsupervised log-based failure detection techniques for model building in this task.

In our experiment, we adopted DeepLog [32] and LogCluster [110] (two typical semi-supervised log-based failure detection techniques) and PCA [33] (a typical unsupervised log-based failure detection technique) as the representatives. DeepLog builds a failure detection model based on a large amount of normal log data via Long Short-Term Memory (LSTM) [50], and then identifies a failure by predicting the next log event and comparing it with the actual one. Since DeepLog predicts the next log event for failure detection, we adapted it to prioritize test cases based on the number of its identified anomalous log events in the log sequence of each test case. LogCluster [110] applies the Agglomerative Hierarchical clustering algorithm [38] to cluster log sequences for failure detection. We adapted it to prioritize test cases by calculating the minimum distance between the log sequence of each test case and the centroids of normal clusters (identified based on training data). PCA [33] projects the log sequence of each test case to the normal space and the anomalous space, which are constructed based on training data, for failure detection. We adapted it to prioritize test cases based on the projection size in the anomalous space.

In this experiment, we used Blueflood, Flume, and Wicket as the representative subjects. On average, the size of training log sequences is about 2,600 for them. Table 11 shows the APFD results of those TCP techniques adapted from the existing log-based failure detection techniques. In this table, we marked the best result as **bold** for each studied subject. From this table, we found that directly adapting the existing log-based failure detection techniques for TCP performs worse than our well-designed LogTCP specific to the task of test case prioritization in terms of APFD and all the bold values belong to the results of LogTCP, which further demonstrates the value of LogTCP. In the future, we may design the log-based failure detection technique specific to the task of test case prioritization, so as to further improve its effectiveness.

6.3 Efficiency of LogTCP

As presented in Section 5, LogTCP achieves better prioritization effectiveness than the existing BTCP techniques and significantly bridges the effectiveness gap between BTCP and WTCP. Here, we further analyzed the time and memory costs of each studied TCP technique in order to better understand the efficiency of LogTCP.

Table 12 presents the average time and memory costs of each studied TCP technique across all the subjects. From this table, in terms of the average time cost, LogTCP$^{arp}_{semantics}$ spends the most time (i.e., 149.76 seconds) among the seven log-based TCP techniques, whose time cost consists of

Table 12. Efficiency comparison among studied TCP techniques in terms of average time and memory usage

Technique	Time (s)	Memory (MB)
$\text{LogTCP}^{total}_{count}$	56.57	119.79
$\text{LogTCP}^{additional}_{count}$	57.55	129.58
$\text{LogTCP}^{arp}_{count}$	56.92	128.41
$\text{LogTCP}^{total}_{ordering}$	122.26	113.14
$\text{LogTCP}^{additional}_{ordering}$	126.86	136.58
$\text{LogTCP}^{arp}_{ordering}$	122.55	161.59
$\text{LogTCP}^{arp}_{semantics}$	149.76	506.58
BTCP_{string}	79.90	448.60
BTCP_{topic}	103.42	244.06
BTCP_{FAST}	0.52	18.70
WTCP_{total}	0.03	109.72
$\text{WTCP}_{additional}$	0.55	118.71
WTCP_{arp}	292.76	470.17
WTCP_{search}	8.37	112.97

log pre-processing time, log representation time, and prioritization time. BTCP_{topic} spends the most time (i.e., 103.42 seconds) among the three existing BTCP techniques, and WTCP_{arp} spends the most time (i.e., 292.76 seconds) among the four existing WTCP techniques. Overall, all the studied techniques spend less 292.76 seconds on test case prioritization on average. In particular, the test case prioritization process is conducted *offline* in regression testing [23, 55, 60, 69, 108], and thus the time costs of all the studied techniques are acceptable in practice.

In terms of the average memory usage, $\text{LogTCP}^{arp}_{semantics}$ consumes the most memory (i.e., 506.58 MB) among the seven log-based TCP techniques, BTCP_{string} consumes the most memory (i.e., 448.60 MB) among the three existing BTCP techniques, and WTCP_{arp} consumes the most memory (i.e., 470.17 MB) among the four existing WTCP techniques. Overall, all the studied techniques consume less 506.58 MB memory on test case prioritization on average, which is also acceptable in practice.

To sum up, the results demonstrate the efficiency of the studied techniques. Although LogTCP spends more time or consumes more memory than some of the existing BTCP or WTCP techniques on average, the cost of LogTCP is still acceptable in terms of both prioritization time and memory usage in practice. Besides, LogTCP just requires raw log messages for test case prioritization, which are easy to collect in a continuous integration environment. That further demonstrates the practicability of LogTCP, i.e., requiring little effort to integrate LogTCP in a continuous integration environment.

6.4 Extension of LogTCP

First, our extensive study has demonstrated that mining test logs is indeed able to improve the effectiveness of BTCP, even achieve competitive effectiveness to WTCP. That shows the great potential of logs in the area of TCP. Currently, LogTCP directly utilizes the logs produced according to the original logging statements in the project under test, but they may be not sufficient for the TCP task. In the future, we could improve the logging practice by suggesting the logging contents and locations specific to regression testing tasks, in order to further improve the effectiveness of LogTCP.

Second, we have proposed several log representation strategies by considering different features in test logs, but these features may be not comprehensive for TCP. Due to the rapid development of deep learning, it could be helpful to incorporate them to automatically and systematically extract features from logs. For example, it could build a log-embedding model based on historical test logs of the project under test. In historical test logs, we can obtain the information of test failures, enabling the feasibility of supervised methods for log-embedding model building. In the literature, supervised methods could be more effective than unsupervised methods (our used log representation strategies in LogTCP are unsupervised) due to incorporating more known information [29, 31, 96].

Third, we used each kind of log feature individually in LogTCP. Actually, these features reflect different aspects of logs, and thus it is likely to integrate them through ensemble learning. Moreover, LogTCP is also orthogonal to other BTCP techniques that mostly rely on the textual information of test cases themselves. Therefore, it may be also helpful to improve the effectiveness of BTCP by integrating various features in different kinds of BTCP techniques.

6.5 Threats to validity

The threats to *internal* validity mainly lie in the implementation of TCP techniques (including our LogTCP techniques and compared techniques) and experimental scripts, and the method of identifying flaky tests. Regarding the compared techniques, we directly adopted the existing implementation for the studied WTCP techniques, which are released by the existing work [70, 126], and re-implemented the compared BTCP techniques based on their descriptions in the corresponding papers [47, 59, 108]. To reduce this kind of threat from implementation, two authors have carefully checked all our code. Also, we have adopted some mature tools to facilitate our implementation as presented in Section 4.4. Regarding flaky tests, we ran each test case several times (i.e., 10 times in our study) for identifying and removing flaky tests following the existing work [7, 78, 102]. Indeed, such a way may not identify and remove all the flaky tests, which may affect the effectiveness of test case prioritization. In the future, we will incorporate more advanced methods to identify flaky tests in order to further reduce this kind of threat.

The threats to *external* validity mainly lie in the subjects, logs, and faults. Although the used subjects in our study may not sufficiently represent other subjects, we have used 10 widely-studied subjects. In particular, we selected these subjects without any subjective bias and these subjects have diverse functionalities. In the future, we will repeat our experiments on more subjects to further reduce this kind of threat. Regarding the kind of threat from logs, the logs in all the studied subjects are produced based on the Log4j or Logback library. However, this kind of threat might be not serious since our LogTCP framework does not rely on the log styles and our used log parser (i.e., Drain3) can process various styles of logs. Regarding the kind of threat from faults, we used mutation faults for evaluation following the existing work [3, 54, 70, 73]. In particular, according to the conclusion from the existing study of investigating the threats of mutant faults [83], we filtered out both duplicate and live mutation faults. In the future, we will try to collect a large number of regression faults on the subjects with logs, in order to further reduce this kind of threat.

The threats to *construct* validity mainly lie in the measurements, regression scenario, randomness, and the inaccuracy from the dependent tools. Following the existing work [70, 73, 114], we adopted both APFD and RAUC-s (with four different settings of s) as the metrics for TCP effectiveness, but they may not represent other metrics. In the future, we will adopt more metrics to more sufficiently measure TCP effectiveness, such as $APFD_c$ [34]. Normalized APFD (NAPFD) is also a metric of measuring the effectiveness of test case prioritization [87]. Although the formulae of RAUC [114] (one metric used in our study) and NAPFD are different, both of them have the same intention. Specifically, both of them consider the scenario where the entire prioritized test suite may be not always executed completely due to the testing time limitation in practice. Therefore, even though

the specific values obtained from them are different, the conclusions are the same in terms of both metrics. Regarding the kind of threat from the regression scenario, we regarded the version without faults as the former version and the version with mutation faults as the current version following the existing TCP studies [23, 69, 73, 122]. To reduce this kind of threat, we constructed 100 faulty versions for each subject, and in the future, we will collect real faults from real-world regression scenarios. In addition, as presented in Section 4.4, we repeated all the TCP techniques involving randomness 5 times and calculated the average results in our study, in order to reduce the kind of threat from randomness. Finally, LogTCP depends on some tools, such as Drain3, and these tools may also bring some inaccuracies. To reduce this kind of threat, we adopted state-of-the-art tools in the corresponding tasks. Moreover, LogTCP is not specific to the currently used tools and can be easily extended by incorporating more advanced tools in the future.

7 RELATED WORK

Our work is related to both test case prioritization and log analysis, and thus we present related work from both aspects.

7.1 Test Case Prioritization

As presented before, TCP can be divided into two categories, i.e., BTCP and WTCP, based on whether the source code information is used. In particular, there are some excellent survey papers on TCP [68, 120].

Besides BTCP$_{string}$, BTCP$_{topic}$, and BTCP$_{FAST}$ presented before, some BTCP techniques were proposed in the context of combinatorial interaction testing (CIT) for highly-configurable systems [11, 26], which prioritize test cases based on CIT coverage (e.g., pair-wise coverage). Also, Rogstad et al. [89] proposed to prioritize test cases based on the diversity of program outputs. Sampath et al. [95] proposed to prioritize user-session-based test cases for web applications testing based on test-case lengths, appearance frequency of request sequences, and systematic coverage of parameter-values and their interactions. Anderson et al. [1] proposed to utilize the identified usage patterns based on telemetry for test case prioritization. Srikanth et al. [106] prioritized building acceptance tests for an enterprise cloud application based on historical service information. In addition, there are some BTCP techniques based on requirement information [49, 56, 107], such as customer-assigned priority, requirement traceability, and the relationship between test cases and risky requirements. These kinds of information are specific to certain categories of software or cannot be always easily available in practice. Different from these existing BTCP techniques, our work is the first to improve the effectiveness of BTCP through log analysis. Moreover, log data are general and tend to be easily obtained in practice. Our experimental results have demonstrated that LogTCP is indeed able to outperform the state-of-the-art BTCP technique.

Besides coverage-based WTCP presented before, there are a number of other WTCP techniques, such as mutation-based techniques [35, 69, 111] and information-retrieval-based techniques [85, 94]. For example, Lou et al. [69] proposed a mutation-based TCP technique in the scenario of software evolution by utilizing mutation faults on the difference between two versions. Peng et al. [85] proposed an enhanced information-retrieval-based TCP technique by incorporating test-case text, code changes, historical test-case execution time, and test-case failure frequencies. Different from WTCP techniques, our work belongs to the category of BTCP but aims to bridge the effectiveness gap between WTCP and BTCP through log analysis. Our experimental results have shown that LogTCP can achieve competitive effectiveness with the state-of-the-art WTCP technique.

There are a number of empirical studies on TCP [74, 82, 84, 103, 116]. For example, Rothermel et al. [91] empirically evaluated the effectiveness of several coverage-based and mutation-based WTCP techniques. Henard et al. [47] conducted an empirical study to compare BTCP and WTCP in

terms of both effectiveness and efficiency. Lu et al. [70] conducted an empirical study to investigate the effectiveness of four representative coverage-based WTCP techniques (that are also used in our study) in the real-world scenario of software evolution. Luo et al. [73] empirically compared five static TCP techniques and four dynamic TCP techniques by considering several factors, such as test case granularity and subject size. Different from these empirical studies, our work conducted an extensive study to explore whether incorporating log analysis can help improve the effectiveness of BTCP and thus bridge the effectiveness gap between WTCP and BTCP.

7.2 Log Analysis

In the literature, there is a great amount of work focusing on log analysis [45], including logging, log parsing, and log mining. Logging aims to improve logging practices, including *what-to-log* [42, 67, 80, 100] (i.e., providing sufficient and concise information in logging statements), *where-to-log* [63, 118, 124, 128] (i.e., determining the proper location of logging statements), and *how-to-log* [15, 62, 99] (i.e., maintaining high-quality logging statements). Log parsing has been introduced before and adopted in our proposed LogTCP. Over the years, many log parsing methods have been proposed, including frequent-pattern-mining based methods (e.g., LFA [81] and Logram [28]), clustering-based methods (e.g., LKE [37] and LogCluster [110]), and heuristics-based methods (e.g., AEL [52] and Drain [43, 44]). In LogTCP, we adopted the widely-used Drain3. After parsing log messages into log events, they are used for a series of subsequent tasks, such as anomaly detection [117, 123], failure prediction [65, 93], and failure diagnosis [112, 121, 127]. Most of those techniques adopted machine learning or deep learning algorithms to build models based on processed log data. In particular, some work on software testing also utilizes logs, e.g., Andrews et al.[2, 4] relieved the test oracle problem using logs and Chen et al. [17] proposed to estimate code coverage measures from logs.

Different from these log analysis work, our work is the first to incorporate log analysis into test case prioritization. Specifically, our work aims to boost BTCP through log analysis.

8 CONCLUSION

Both white-box test case prioritization (WTCP) and black-box test case prioritization (BTCP) suffer from limitations in their practical use. The former relies on source code information, which can achieve great prioritization effectiveness but cannot be applicable in many practical scenarios (where source code is unavailable), while the latter gets rid of the limitation of requiring source code information, but tends to perform worse than WTCP due to less information used for TCP. To promote the practicability of TCP, in this work we explore better BTCP to bridge the effectiveness gap between BTCP and WTCP through log analysis. Specifically, we first design a log-based TCP framework (called LogTCP), including log pre-processing, log representation, and test case prioritization components. Then, we conduct an empirical study to investigate the effectiveness of LogTCP based on 10 diverse Java projects from GitHub. The results demonstrate that LogTCP significantly performs better than the state-of-the-art BTCP technique, even achieves competitive effectiveness to the state-of-the-art WTCP technique in average fault detection. In particular, we recommend the LogTCP technique combining the semantics-based or ordering-based log representation strategy with the adaptive random prioritization strategy as the first choice in practice due to its better effectiveness.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (Grant No. 62002256) and Fund projects in the technical field of the foundation strengthening plan (2020-JCJQ-JJ-490).

REFERENCES

[1] Jeff Anderson, Maral Azizi, Saeed Salem, and Hyunsook Do. 2019. On the use of usage patterns from telemetry data for test case prioritization. *Information and Software Technology* 113 (2019), 110–130.

[2] James H Andrews. 1998. Testing using log file analysis: tools, methods, and issues. In *Proceedings 13th IEEE International Conference on Automated Software Engineering*. IEEE, 157–166.

[3] James H Andrews, Lionel C Briand, and Yvan Labiche. 2005. Is mutation an appropriate tool for testing experiments?. In *Proceedings of the 27th International Conference on Software Engineering*. ACM, 402–411.

[4] James H Andrews and Yingjun Zhang. 2003. General test result checking with log file analysis. *IEEE Transactions on Software Engineering* 29, 7 (2003), 634–648.

[5] Han Anu, Jie Chen, Wenchang Shi, Jianwei Hou, Bin Liang, and Bo Qin. 2019. An approach to recommendation of verbosity log levels based on logging intention. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 125–134.

[6] Andrea Arcuri and Lionel C. Briand. 2011. Adaptive random testing: an illusion of effectiveness?. In *International Symposium on Software Testing and Analysis*. ACM, 265–275.

[7] Jonathan Bell, Owolabi Legunsen, Michael Hilton, Lamyaa Eloussi, Tifany Yung, and Darko Marinov. 2018. DeFlaker: Automatically detecting flaky tests. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 433–444.

[8] Yoav Benjamini and Yosef Hochberg. 1995. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B (Methodological)* 57, 1 (1995), 289–300.

[9] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *the Journal of machine Learning research* 3 (2003), 993–1022.

[10] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146.

[11] Renée C Bryce and Charles J Colbourn. 2006. Prioritized interaction testing for pair-wise coverage with seeding and constraints. *Information and Software Technology* 48, 10 (2006), 960–970.

[12] Taejoon Byun, Vaibhav Sharma, Abhishek Vijayakumar, Sanjai Rayadurgam, and Darren Cofer. 2019. Input prioritization for testing neural networks. In *2019 IEEE International Conference On Artificial Intelligence Testing*. IEEE, 63–70.

[13] William B Cavnar and John M Trenkle. 1994. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*, Vol. 161175. Citeseer.

[14] Kwok Ping Chan, Tsong Yueh Chen, and Dave Towey. 2006. Restricted random testing: Adaptive random testing by exclusion. *International Journal of Software Engineering and Knowledge Engineering* 16, 04 (2006), 553–584.

[15] Boyuan Chen and Zhen Ming Jiang. 2017. Characterizing and detecting anti-patterns in the logging code. In *2017 IEEE/ACM 39th International Conference on Software Engineering*. IEEE, 71–81.

[16] Boyuan Chen and Zhen Ming Jack Jiang. 2017. Characterizing logging practices in java-based open source software projects–a replication study in apache software foundation. *Empirical Software Engineering* 22, 1 (2017), 330–374.

[17] Boyuan Chen, Jian Song, Peng Xu, Xing Hu, and Zhen Ming Jiang. 2018. An automated approach to estimating code coverage measures via execution logs. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 305–316.

[18] Junjie Chen. 2018. Learning to accelerate compiler testing. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. 472–475.

[19] Junjie Chen, Yanwei Bai, Dan Hao, Yingfei Xiong, Hongyu Zhang, and Bing Xie. 2017. Learning to prioritize test programs for compiler testing. In *2017 IEEE/ACM 39th International Conference on Software Engineering*. IEEE, 700–711.

[20] Junjie Chen, Yanwei Bai, Dan Hao, Yingfei Xiong, Hongyu Zhang, Lu Zhang, and Bing Xie. 2016. Test case prioritization for compilers: A text-vector based approach. In *2016 IEEE international conference on software testing, verification and validation (ICST)*. IEEE, 266–277.

[21] Junjie Chen, Yanwei Bai, Dan Hao, Lingming Zhang, Lu Zhang, and Bing Xie. 2017. How do assertions impact coverage-based test-suite reduction?. In *2017 IEEE International Conference on Software Testing, Verification and Validation*. IEEE, 418–423.

[22] Junjie Chen, Yanwei Bai, Dan Hao, Lingming Zhang, Lu Zhang, Bing Xie, and Hong Mei. 2016. Supporting oracle construction via static analysis. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 178–189.

[23] Junjie Chen, Yiling Lou, Lingming Zhang, Jianyi Zhou, Xiaoleng Wang, Dan Hao, and Lu Zhang. 2018. Optimizing test prioritization via test distribution analysis. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 656–667.

[24] Junjie Chen, Guancheng Wang, Dan Hao, Yingfei Xiong, Hongyu Zhang, Lu Zhang, and Bing Xie. 2018. Coverage prediction for accelerating compiler testing. *IEEE Transactions on Software Engineering* 47, 2 (2018), 261–278.

[25] Tsong Yueh Chen, Hing Leung, and Ieng Kei Mak. 2004. Adaptive random testing. In *Annual Asian Computing Science Conference*. Springer, 320–329.
[26] Myra B Cohen, Matthew B Dwyer, and Jiangfan Shi. 2008. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Transactions on Software Engineering* 34, 5 (2008), 633–650.
[27] Emilio Cruciani, Breno Miranda, Roberto Verdecchia, and Antonia Bertolino. 2019. Scalable approaches for test suite reduction. In *2019 IEEE/ACM 41st International Conference on Software Engineering*. IEEE, 419–429.
[28] Hetong Dai, Heng Li, Che Shao Chen, Weiyi Shang, and Tse-Hsun Chen. 2020. Logram: Efficient log parsing using n-gram dictionaries. *IEEE Transactions on Software Engineering* abs/2001.03038 (2020).
[29] Rajashree Dash, Rajib Lochan Paramguru, and Rasmita Dash. 2011. Comparative analysis of supervised and unsupervised discretization techniques. *International Journal of Advances in Science and Technology* 2, 3 (2011), 29–37.
[30] Bogdan Dit, Latifa Guerrouj, Denys Poshyvanyk, and Giuliano Antoniol. 2011. Can better identifier splitting techniques help feature location?. In *2011 IEEE 19th International Conference on Program Comprehension*. IEEE, 11–20.
[31] James Dougherty, Ron Kohavi, and Mehran Sahami. 1995. Supervised and unsupervised discretization of continuous features. In *Machine learning proceedings 1995*. Elsevier, 194–202.
[32] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1285–1298.
[33] Ricardo Dunia and S Joe Qin. 1997. Multi-dimensional fault diagnosis using a subspace approach. In *American Control Conference*, Vol. 5. Citeseer.
[34] Sebastian Elbaum, Alexey Malishevsky, and Gregg Rothermel. 2001. Incorporating varying test costs and fault severities into test case prioritization. In *Proceedings of the 23rd International Conference on Software Engineering*. IEEE, 329–338.
[35] Sebastian Elbaum, Alexey G Malishevsky, and Gregg Rothermel. 2002. Test case prioritization: A family of empirical studies. *IEEE Transactions on Software Engineering* 28, 2 (2002), 159–182.
[36] Gordon Fraser and Andrea Arcuri. 2011. Evosuite: automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 416–419.
[37] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining*. IEEE, 149–158.
[38] John C Gower and Gavin JS Ross. 1969. Minimum spanning trees and single linkage cluster analysis. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 18, 1 (1969), 54–64.
[39] Haixuan Guo, Shuhan Yuan, and Xintao Wu. 2021. Logbert: Log anomaly detection via bert. In *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.
[40] Jiawei Han, Jian Pei, and Micheline Kamber. 2011. *Data mining: concepts and techniques*. Elsevier.
[41] Dan Hao, Lu Zhang, and Hong Mei. 2016. Test-case prioritization: achievements and challenges. *Frontiers of Computer Science* 10, 5 (2016), 769–777.
[42] Pinjia He, Zhuangbin Chen, Shilin He, and Michael R Lyu. 2018. Characterizing the natural language descriptions in software logging statements. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 178–189.
[43] Pinjia He, Jieming Zhu, Pengcheng Xu, Zibin Zheng, and Michael R Lyu. 2018. A directed acyclic graph approach to online log parsing. *arXiv preprint arXiv:1806.04356* (2018).
[44] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services*. IEEE, 33–40.
[45] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R Lyu. 2021. A survey on automated log analysis for reliability engineering. *Comput. Surveys* 54, 6 (2021), 1–37.
[46] Hadi Hemmati, Andrea Arcuri, and Lionel Briand. 2013. Achieving scalable model-based testing through test case diversity. *ACM Transactions on Software Engineering and Methodology* 22, 1 (2013), 1–42.
[47] Christopher Henard, Mike Papadakis, Mark Harman, Yue Jia, and Yves Le Traon. 2016. Comparing white-box and black-box test prioritization. In *2016 IEEE/ACM 38th International Conference on Software Engineering*. IEEE, 523–534.
[48] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, and Yves Le Traon. 2013. Assessing software product line testing via model-based mutation: An application to similarity testing. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation workshops*. IEEE, 188–197.
[49] Charitha Hettiarachchi, Hyunsook Do, and Byoungju Choi. 2014. Effective regression testing using requirements and risks. In *2014 Eighth International Conference on Software Security and Reliability (SERE)*. IEEE, 157–166.
[50] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[51] Bo Jiang, Zhenyu Zhang, Wing Kwong Chan, and TH Tse. 2009. Adaptive random test case prioritization. In *2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 233–244.

[52] Zhen Ming Jiang, Ahmed E Hassan, Parminder Flora, and Gilbert Hamann. 2008. Abstracting execution logs to execution events for enterprise applications (short paper). In *2008 The Eighth International Conference on Quality Software*. IEEE, 181–186.

[53] René Just, Darioush Jalali, and Michael D Ernst. 2014. 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*. 437–440.

[54] René Just, Darioush Jalali, Laura Inozemtseva, Michael D Ernst, Reid Holmes, and Gordon Fraser. 2014. Are mutants a valid substitute for real faults in software testing?. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 654–665.

[55] Jung-Min Kim and Adam Porter. 2002. A history-based test prioritization technique for regression testing in resource constrained environments. In *Proceedings of the 24th international conference on software engineering*. 119–129.

[56] R Krishnamoorthi and SA Sahaaya Arul Mary. 2009. Factor oriented requirement coverage based system test case prioritization of new and regression test cases. *Information and Software Technology* 51, 4 (2009), 799–808.

[57] Jung-Hyun Kwon, In-Young Ko, Gregg Rothermel, and Matt Staats. 2014. Test case prioritization based on information retrieval concepts. In *2014 21st Asia-Pacific Software Engineering Conference*, Vol. 1. IEEE, 19–26.

[58] Wing Lam, Stefan Winter, Angello Astorga, Victoria Stodden, and Darko Marinov. 2020. Understanding reproducibility and characteristics of flaky tests through test reruns in Java projects. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 403–413.

[59] Yves Ledru, Alexandre Petrenko, and Sergiy Boroday. 2009. Using string distances for test case prioritisation. In *2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 510–514.

[60] Yves Ledru, Alexandre Petrenko, Sergiy Boroday, and Nadine Mandran. 2012. Prioritizing test cases with string distances. *Automated Software Engineering* 19, 1 (2012), 65–95.

[61] Heng Li, Weiyi Shang, and Ahmed E Hassan. 2017. Which log level should developers choose for a new logging statement? *Empirical Software Engineering* 22, 4 (2017), 1684–1716.

[62] Shanshan Li, Xu Niu, Zhouyang Jia, Xiangke Liao, Ji Wang, and Tao Li. 2020. Guiding log revisions by learning from software evolution history. *Empirical Software Engineering* 25, 3 (2020), 2302–2340.

[63] Zhenhao Li, Tse-Hsun Chen, and Weiyi Shang. 2020. Where shall we log? studying and suggesting logging locations in code blocks. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 361–372.

[64] Zheng Li, Mark Harman, and Robert M Hierons. 2007. Search algorithms for regression test case prioritization. *IEEE Transactions on software engineering* 33, 4 (2007), 225–237.

[65] Qingwei Lin, Ken Hsieh, Yingnong Dang, Hongyu Zhang, Kaixin Sui, Yong Xu, Jian-Guang Lou, Chenggang Li, Youjiang Wu, Randolph Yao, Murali Chintalapati, and Dongmei Zhang. 2018. Predicting node failure in cloud service systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 480–490.

[66] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log clustering based problem identification for online service systems. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion*. IEEE, 102–111.

[67] Zhongxin Liu, Xin Xia, David Lo, Zhenchang Xing, Ahmed E. Hassan, and Shanping Li. 2021. Which Variables Should I Log? *IEEE Transactions on Software Engineering* (2021), 2012–2031.

[68] Yiling Lou, Junjie Chen, Lingming Zhang, and Dan Hao. 2019. A survey on regression test-case prioritization. In *Advances in Computers*. Vol. 113. Elsevier, 1–46.

[69] Yiling Lou, Dan Hao, and Lu Zhang. 2015. Mutation-based test-case prioritization in software evolution. In *2015 IEEE 26th International Symposium on Software Reliability Engineering*. IEEE, 46–57.

[70] Yafeng Lu, Yiling Lou, Shiyang Cheng, Lingming Zhang, Dan Hao, Yangfan Zhou, and Lu Zhang. 2016. How does regression test prioritization perform in real-world software evolution?. In *Proceedings of the 38th International Conference on Software Engineering*. 535–546.

[71] Qi Luo, Kevin Moran, and Denys Poshyvanyk. 2016. A large-scale empirical comparison of static and dynamic test case prioritization techniques. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 559–570.

[72] Qi Luo, Kevin Moran, Denys Poshyvanyk, and Massimiliano Di Penta. 2018. Assessing test case prioritization on real faults and mutants. In *2018 IEEE international conference on software maintenance and evolution*. IEEE, 240–251.

[73] Qi Luo, Kevin Moran, Lingming Zhang, and Denys Poshyvanyk. 2018. How do static and dynamic test case prioritization techniques perform on modern software systems? An extensive study on GitHub projects. *IEEE Transactions on Software Engineering* 45, 11 (2018), 1054–1080.

[74] Mostafa Mahdieh, Seyed-Hassan Mirian-Hosseinabadi, Khashayar Etemadi, Ali Nosrati, and Sajad Jalali. 2020. Incorporating fault-proneness estimations into coverage-based test case prioritization methods. *Information and Software Technology* 121 (2020), 106269.
[75] Alexey G Malishevsky, Gregg Rothermel, and Sebastian Elbaum. 2002. Modeling the cost-benefits tradeoffs for regression testing techniques. In *International Conference on Software Maintenance, 2002. Proceedings.* IEEE, 204–213.
[76] Dusica Marijan, Arnaud Gotlieb, and Sagar Sen. 2013. Test case prioritization for continuous regression testing: An industrial case study. In *2013 IEEE International Conference on Software Maintenance.* IEEE, 540–543.
[77] Hong Mei, Dan Hao, Lingming Zhang, Lu Zhang, Ji Zhou, and Gregg Rothermel. 2012. A static approach to prioritizing junit test cases. *IEEE Transactions on Software Engineering* 38, 6 (2012), 1258–1275.
[78] John Micco. 2017. The state of continuous integration testing@ Google. (2017).
[79] Breno Miranda, Emilio Cruciani, Roberto Verdecchia, and Antonia Bertolino. 2018. FAST approaches to scalable similarity-based test case prioritization. In *2018 IEEE/ACM 40th International Conference on Software Engineering.* IEEE, 222–232.
[80] Tsuyoshi Mizouchi, Kazumasa Shimari, Takashi Ishio, and Katsuro Inoue. 2019. PADLA: a dynamic log level adapter using online phase detection. In *2019 IEEE/ACM 27th International Conference on Program Comprehension.* IEEE, 135–138.
[81] Meiyappan Nagappan and Mladen A Vouk. 2010. Abstracting log lines to log event types for mining software system logs. In *2010 7th IEEE Working Conference on Mining Software Repositories.* IEEE, 114–117.
[82] Manoj Kumar Pachariya. 2020. Building Ant System for Multi-Faceted Test Case Prioritization: An Empirical Study. *International Journal of Software Innovation* 8, 2 (2020), 23–37.
[83] Mike Papadakis, Christopher Henard, Mark Harman, Yue Jia, and Yves Le Traon. 2016. Threats to the validity of mutation-based test assessment. In *Proceedings of the 25th International Symposium on Software Testing and Analysis.* 354–365.
[84] David Paterson, José Campos, Rui Abreu, Gregory M Kapfhammer, Gordon Fraser, and Phil McMinn. 2019. An empirical study on the use of defect prediction for test case prioritization. In *2019 12th IEEE Conference on Software Testing, Validation and Verification.* IEEE, 346–357.
[85] Qianyang Peng, August Shi, and Lingming Zhang. 2020. Empirically revisiting and enhancing IR-based test-case prioritization. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis.* 324–336.
[86] Adithya Abraham Philip, Ranjita Bhagwan, Rahul Kumar, Chandra Sekhar Maddila, and Nachiappan Nagppan. 2019. FastLane: Test minimization for rapidly deployed large-scale online services. In *2019 IEEE/ACM 41st International Conference on Software Engineering.* IEEE, 408–418.
[87] Xiao Qu, Myra B Cohen, and Katherine M Woolf. 2007. Combinatorial interaction regression testing: A study of test case generation and prioritization. In *2007 IEEE International Conference on Software Maintenance.* IEEE, 255–264.
[88] Anand Rajaraman and Jeffrey David Ullman. 2011. *Mining of massive datasets.* Cambridge University Press.
[89] Erik Rogstad, Lionel Briand, and Richard Torkar. 2013. Test case selection for black-box regression testing of database applications. *Information and Software Technology* 55, 10 (2013), 1781–1795.
[90] Gregg Rothermel and Mary Jean Harrold. 1997. A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology* 6, 2 (1997), 173–210.
[91] Gregg Rothermel, Roland H Untch, Chengyun Chu, and Mary Jean Harrold. 1999. Test case prioritization: An empirical study. In *Proceedings IEEE International Conference on Software Maintenance-1999.'Software Maintenance for Business Change'(Cat. No. 99CB36360).* IEEE, 179–188.
[92] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. 2001. Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering* 27, 10 (2001), 929–948.
[93] Barbara Russo, Giancarlo Succi, and Witold Pedrycz. 2015. Mining system logs to learn error predictors: a case study of a telemetry system. *Empirical Software Engineering* 20, 4 (2015), 879–927.
[94] Ripon K Saha, Lingming Zhang, Sarfraz Khurshid, and Dewayne E Perry. 2015. An information retrieval approach for regression test prioritization based on program changes. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 268–279.
[95] Sreedevi Sampath, Renee C Bryce, Gokulanand Viswanath, Vani Kandimalla, and A Gunes Koru. 2008. Prioritizing user-session-based test cases for web applications testing. In *2008 1st International Conference on Software Testing, Verification, and Validation.* IEEE, 141–150.
[96] Ramadass Sathya and Annamma Abraham. 2013. Comparison of supervised and unsupervised learning algorithms for pattern classification. *International Journal of Advanced Research in Artificial Intelligence* 2, 2 (2013), 34–38.
[97] Juliet Popper Shaffer. 1995. Multiple hypothesis testing. *Annual review of psychology* 46, 1 (1995), 561–584.
[98] Hina Shah, Saurabh Sinha, and Mary Jean Harrold. 2011. Outsourced, offshored software-testing practice: Vendor-side experiences. In *2011 IEEE sixth International Conference on Global Software Engineering.* IEEE, 131–140.

[99] Weiyi Shang, Meiyappan Nagappan, and Ahmed E Hassan. 2015. Studying the relationship between logging characteristics and the code quality of platform software. *Empirical Software Engineering* 20, 1 (2015), 1–27.
[100] Weiyi Shang, Meiyappan Nagappan, Ahmed E Hassan, and Zhen Ming Jiang. 2014. Understanding log lines using development knowledge. In *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 21–30.
[101] Samuel Sanford Shapiro and Martin B Wilk. 1965. An analysis of variance test for normality (complete samples). *Biometrika* 52, 3/4 (1965), 591–611.
[102] August Shi, Wing Lam, Reed Oei, Tao Xie, and Darko Marinov. 2019. iFixFlakies: A framework for automatically fixing order-dependent flaky tests. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 545–555.
[103] Donghwan Shin, Shin Yoo, Mike Papadakis, and Doo-Hwan Bae. 2019. Empirical evaluation of mutation-based test case prioritization techniques. *Software Testing, Verification and Reliability* 29, 1-2 (2019), e1695.
[104] Dennis Silva, Ricardo Rabelo, Matheus Campanha, Pedro Santos Neto, Pedro Almir Oliveira, and Ricardo Britto. 2016. A hybrid approach for test case prioritization and selection. In *2016 IEEE Congress on Evolutionary Computation*. IEEE, 4508–4515.
[105] Charles Spearman. 1961. The proof and measurement of association between two things. (1961).
[106] Hema Srikanth, Mikaela Cashman, and Myra B Cohen. 2016. Test case prioritization of build acceptance tests for an enterprise cloud application: An industrial case study. *Journal of Systems and Software* 119 (2016), 122–135.
[107] Hema Srikanth, Laurie Williams, and Jason Osborne. 2005. System test case prioritization of new and regression test cases. In *2005 International Symposium on Empirical Software Engineering, 2005.* IEEE, 10–pp.
[108] Stephen W Thomas, Hadi Hemmati, Ahmed E Hassan, and Dorothea Blostein. 2014. Static test case prioritization using topic models. *Empirical Software Engineering* 19, 1 (2014), 182–212.
[109] Zhao Tian, Junjie Chen, Qihao Zhu, Junjie Yang, and Lingming Zhang. 2022. Learning to Construct Better Mutation Faults. In *37th IEEE/ACM International Conference on Automated Software Engineering*. to appear.
[110] Risto Vaarandi and Mauno Pihelgas. 2015. Logcluster-a data clustering and pattern mining algorithm for event logs. In *2015 11th International Conference on Network and Service Management*. IEEE, 1–7.
[111] Jeffrey M. Voas. 1992. PIE: A dynamic failure-based technique. *IEEE Transactions on Software Engineering* 18, 8 (1992), 717.
[112] Lingzhi Wang, Nengwen Zhao, Junjie Chen, Pinnong Li, Wenchi Zhang, and Kaixin Sui. 2020. Root-cause metric location for microservice systems via log anomaly detection. In *2020 IEEE International Conference on Web Services*. IEEE, 142–150.
[113] Song Wang, Jaechang Nam, and Lin Tan. 2017. QTEP: Quality-aware test case prioritization. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 523–534.
[114] Zan Wang, Hanmo You, Junjie Chen, Yingyi Zhang, Xuyuan Dong, and Wenbin Zhang. 2021. Prioritizing Test Inputs for Deep Neural Networks via Mutation Analysis. In *2021 IEEE/ACM 43rd International Conference on Software Engineering*. IEEE, 397–409.
[115] Robert F Woolson. 2007. Wilcoxon signed-rank test. *Wiley Encyclopedia of Clinical Trials* (2007), 1–3.
[116] Lei Xiao, Huaikou Miao, Weiwei Zhuang, and Shaojun Chen. 2017. An empirical study on clustering approach combining fault prediction for test case prioritization. In *2017 IEEE/ACIS 16th International Conference on Computer and Information Science*. IEEE, 815–820.
[117] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. 2021. Semi-supervised log-based anomaly detection via probabilistic label estimation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering*. IEEE, 1448–1460.
[118] Kundi Yao, Guilherme B de Padua, Weiyi Shang, Catalin Sporea, Andrei Toma, and Sarah Sajedi. 2020. Log4Perf: Suggesting and updating logging locations for web-based systems' performance monitoring. *Empirical Software Engineering* 25, 1 (2020), 488–531.
[119] Shin Yoo and Mark Harman. 2010. Using hybrid algorithm for Pareto efficient multi-objective test suite minimisation. *Journal of Systems and Software* 83 (2010), 689–701.
[120] Shin Yoo and Mark Harman. 2012. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification & Reliability* 22 (2012), 67–120.
[121] Ding Yuan, Haohui Mai, Weiwei Xiong, Lin Tan, Yuanyuan Zhou, and Shankar Pasupathy. 2010. Sherlog: error diagnosis by connecting clues from run-time logs. In *Proceedings of the fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems*. 143–154.
[122] Lingming Zhang, Dan Hao, Lu Zhang, Gregg Rothermel, and Hong Mei. 2013. Bridging the gap between the total and additional test-case prioritization strategies. In *2013 35th International Conference on Software Engineering*. IEEE, 192–201.
[123] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, Junjie Chen, Xiaoting He, Randolph Yao, Jian-Guang Lou, Murali Chintalapati, Furao Shen, and Dongmei

Zhang. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 807–817.

[124] Xu Zhao, Kirk Rodrigues, Yu Luo, Michael Stumm, Ding Yuan, and Yuanyuan Zhou. 2017. Log20: Fully automated optimal placement of log printing statements under specified overhead threshold. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 565–581.

[125] Bo Zhou, Hiroyuki Okamura, and Tadashi Dohi. 2011. Enhancing performance of random testing through Markov chain Monte Carlo methods. *IEEE Trans. Comput.* 62, 1 (2011), 186–192.

[126] Jianyi Zhou, Junjie Chen, and Dan Hao. 2021. Parallel Test Prioritization. *ACM Transactions on Software Engineering and Methodology* 31, 1 (2021), 1–50.

[127] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Dewei Liu, Qilin Xiang, and Chuan He. 2019. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 683–694.

[128] Jieming Zhu, Pinjia He, Qiang Fu, Hongyu Zhang, Michael R Lyu, and Dongmei Zhang. 2015. Learning to log: Helping developers make informed logging decisions. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 415–425.

[129] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. 2019. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice*. IEEE, 121–130.