Received: 30 October 2020 | Revised: 14 September 2021 | Accepted: 20 October 2021

DOI: 10.1002/smr.2409

SPECIAL ISSUE - TECHNOLOGY PAPER

Journal of Software: Evolution and Process WILEY

Sparse reward for reinforcement learning-based continuous integration testing

Yang Yang | Zheng Li | Ying Shang | Qianyu Li

College of Information Science and Technology, Beijing University of Chemical Technology, Beijing, China

Correspondence
Zheng Li, College of Information Science and Technology, Beijing University of Chemical Technology, North Third Ring Road 15, Chaoyang District, Beijing 100029, China.
Email: lizheng@mail.buct.edu.cn

Funding information
National Natural Science Foundation of China, Grant/Award Number: 61872026; 61902015; 62077003; Fundamental Research Funds for the Central Universities, Grant/Award Number: XK1802-4

Abstract

Reinforcement learning (RL) has been used to optimize the continuous integration (CI) testing, where the reward plays a key role in directing the adjustment of the test case prioritization (TCP) strategy. In CI testing, the frequency of integration is usually very high, while the failure rate of test cases is low. Consequently, RL will get scarce rewards in CI testing, which may lead to low learning efficiency of RL and even difficulty in convergence. This paper introduces three rewards to tackle the issue of sparse rewards of RL in CI testing. First, the historical failure density-based reward (*HFD*) is defined, which objectively represents the sparse reward problem. Second, the average failure position-based reward (*AFP*) is proposed to increase the reward value and reduce the impact of sparse rewards. Furthermore, a technique based on additional reward is proposed, which extracts the test occurrence frequency of passed test cases for additional rewards. Empirical studies are conducted on 14 real industry data sets. The experiment results are promising, especially the reward with additional reward can improve NAPFD (Normalized Average Percentage of Faults Detected) by up to 21.97%, enhance Recall with a maximum of 21.87%, and increase TTF (Test to Fail) by an average of 9.99 positions.

KEYWORDS
continuous integration, reinforcement learning, sparse reward, test case prioritization

1 | INTRODUCTION

Continuous integration (CI) is a development model for the environment with high demand, high investment, and high competition.[1] The traditional software development pattern does not adapt to constant change and maintenance,[2] which is hard to guarantee the software quality of the CI system. CI is a cost-effective software development practice with real-time deployment. Developers need to integrate their work in time and provide the corresponding test feedback quickly.[3] Therefore, it is necessary to report the detected faults immediately at any time.[4,5] Compared with the traditional development model, the integration frequency of CI is very high, usually once a day, sometimes even multiple times a day, which makes traditional testing techniques unsuitable for CI testing.[6,7]

Test case prioritization (TCP)[8] prioritizes the execution of the potential failure test cases, which can help CI testing to detect failures as soon as possible. The traditional regression TCP technology is mainly based on various code coverage indicators,[9] where the continuous code analysis for each integration cannot meet the needs of the rapid feedback in CI testing. In addition, the effectiveness of traditional TCP technology depends on the specific program, test cases, and modifications, which does not apply to different programs or modified versions.[10] Therefore, the traditional TCP technology is not suitable for prioritizing the test cases in CI testing.

The TCP in the CI testing is a continuous decision-making problem, where the test cases are continuously reordered for each integration. Spieker et al[11] applied reinforcement learning (RL) to solve the TCP problem in CI testing. RL continuously learns to obtain the best strategy through the interaction between the environment and the subject, which uses the reward as feedback from the environment.[12] The reward is the core part of RL, which will affect the learning effect of RL.[12]

The reward is the feature extracted from the test case, which measures the fault detection ability of the test cases to achieve priority ranking. Spieker et al[11] rewarded the test cases with the current execution results, where the test case failure reward(*TF*) is proposed. They proved the effectiveness of the RL method in the TCP of CI testing, which was compared with heuristic methods. Yang et al[13] further proposed the historical information-based rewards, which extracted the features from the test case's historical information sequence. The historical information-based rewards, especially the Average Percentage of Historical Failure (*APHF*), can achieve a better TCP effect in RL-based CI testing.[11]

However, there are high-frequency integration and low-failure test in CI testing. In RL-based CI testing, frequent integration causes a sharp increase in the historical information of test cases, which increases the computational burden of rewards. With the close correlation between failure test and test failure, the time window-based reward[14] was proposed to select the recent historical information to realize the reward calculation. The low-failure test is common in industrial practice, where most test executions are not rewarded in RL-based CI testing. This is the sparse reward problem of RL. The existence of sparse rewards will lead to slow learning or even difficulty to converge.[15] For example, in the Google open-source data 'GSDTSR',[11] there are only 0.25% failure executions in 1,260,618 test executions, and 99.75% test executions failed to obtain effective rewards in the RL-based CI testing. The time window-based reward[14] further exacerbates the sparse reward.

RL achieves the ultimate goal through the balance of exploration and utilization.[16] The solution to the sparse reward problem of RL mainly involves the improvement of exploration[17](deepening the experience replay of the agent) and the modification of utilization[18] (the improvement of reward). Since reward plays a vital role in RL, this article deals with sparse rewards of RL-based CI testing from the perspective of rewards.

First, due to the scarce failure information, the historical information-based reward is usually very low, which correspondingly leads to sparse rewards. Thence, it is necessary to redefine the reward so that the test case can obtain a higher reward value. The sparsity of historical failures will lead to sparse rewards. Therefore, the historical failure density (*HFD*) is defined as the reward that represents the reward's sparsity, objectively. Further considering the correlation between failure test and test failure, the average failure position (*AFP*) is proposed to measure the average distance between the historical failure executions and the current test, which can greatly increase the reward value.

Second, the low-failure test leads to sparse reward objects, because only failure test cases can be rewarded in RL-based CI testing. In this case, we propose the additional reward to assist the reward objects. According to the requirements of CI testing, testers believe that the test cases submitted in each cycle are necessary.[19] Once the execution fails, system faults can be detected. Therefore, if the reward object is measured with 'important test cases', the passed test cases can be rewarded as additional reward objects. In this article, the test occurrence frequency (*TOF*) is proposed to measure the importance of test cases, including pass and failure test cases. Reward based on *TOF* is defined as a reward with additional reward. *TOF* can be combined with any reward to reduce reward sparsity by increasing the number of rewarded test cases.

The main contributions of this paper include the following:

- This paper first tackles the sparse rewards of RL based on CI testing.
- To reduce reward sparsity, historical failure density is defined to objectively describe the sparse rewards. According to the historical failure distribution, the average failure position is proposed as a reward to increase the reward value.
- In order to reduce the sparse rewards, the test occurrence frequency is used as an additional reward, which can be combined with any other rewards to increase reward objects.
- Empirical studies are carried on 14 real-world industrial programs to evaluate the improved effect of reward sparsity.

This paper is based on the previous conference version, 'Occurrence Frequency and All Historical Failure Information Based Method for TCP in CI',[20] which had been published on the ICSSP 2020. In the revised journal version, (1) the motivation of the paper has been refined, where the problem of reward sparsity is proposed for RL-based CI testing; (2) the rewards proposed have been refined, and the additional reward is formalized in the journal version; (3) the experiments have been scaled up from 5 to 14 subjects, all from industry data sets. Furthermore, the *APHF* reward is added in the experiments as a comparative method, which is proved to be the best historical information-based reward;[13,14] (4) three additional evaluation metrics are added in the experiments, that is, Recall, TTF, and Time Consumption.

The paper is organized as follows: Section 2 summarizes the background and related work. Section 3 presents the sparse reward solutions based on reward design. Section 4 experimentally verifies the effectiveness of the sparse reward solution. Section 5 concludes the article.

2 | BACKGROUND AND RELATED WORK

2.1 | Test case prioritization

Test Case Prioritization (TCP) is a test case optimization method that prioritizes potential failure test cases to find bugs and fix vulnerabilities earlier.[8] TCP technique is used to evaluate and rank individual test cases based on a certain evaluation criteria, which includes coverage-based,[21] modification-based,[22] fault-based,[23] requirement-based,[24] and history-based[25] criteria.

For the complicated prioritization problems, the search-based TCP technology was proposed.[26] Further studies were conducted on the optimization of search algorithms for multi-objective TCP problems,[27] and the GPU parallel technology[28] was proposed to improve search efficiency. Considering the similarity among test cases, Yu et al[29] and Souza et al[30] used a search-based approach based on the similarity among test cases to do TCP.

Many of the existing TCP techniques use code-related information to compute the prioritization criterion, which is based on the consideration that the test case related to the changed codes is more likely to detect faults. For each modification of the CI system, it will take a lot of time to analyze the code-related information. Therefore, this technology is not suitable for CI testing. The CI software scale is large, code coverage analysis is complicated, and resource cost and time cost are high. In 2012, Ledru et al[31] proposed the TCP technique, which used string distance to compare the similarity among test cases. This approach assumes that similar test cases may detect the same faults. They placed the less similar test cases in front of the test sequence to detect more faults. Due to the huge number of test cases in the CI environment, the similarity among test cases cannot be analyzed in real time; thus, such techniques are difficult to be applied in CI testing.

In 2002, Kim and Porter[32] proposed the TCP technique based on historical failure information. The historical failure information-based prioritization method can effectively coordinate the relationship between test cost and fault detection rate, with time and cost saved. In 2013, based on the historical execution information, Marijan et al[33] added execution time of test cases and heuristic factors in specific fields. They calculated the priority of the test cases based on the weighted results of the influence factors. In 2014, Elbaum et al[34] proposed a time window to track the recent historical failure information of the test cases. In 2016, Srikanth and Williams[35] used the idea of a historical window to drive the prioritization of test cases. With the same principle, Strandberg et al[36] proposed a new method to determine the effective ordering of test cases. This method used the expected or observed test case duration, the historical failure information of the test case, and the time interval since the test case's last execution to generate an efficient order of test cases.

The above TCP technologies do not consider the test environment and system cycle, where the configuration requires a lot of time to adjust to adapt to the version changes.[37] However, with the continuous integration of CI systems, the environment and configuration are keep changing, which demand to add or reduce test cases. Experiments have shown that test suite augmentation in traditional methods hampers their effectiveness significantly, whereas source code changes alone do not influence their effectiveness much.[38]

2.2 | Reinforcement learning-based continuous integration testing

As a branch of regression testing, CI testing is more stringent in terms of feedback time and program complexity. The test cases test the continuous modification of CI to ensure that the CI system will work normally. The key to CI is that newly submitted code must pass automatic regression testing before being integrated into the trunk. With the continuous integration of CI systems, real-time testing is required for each integration. Failure test cases are prioritized to detect system failures earlier. In the next CI integration, the priority of the corresponding test cases needs to be re-determined, which is a continuous decision-making process. RL solves the continuous decision-making problem through the interaction between the environment and the agent and adjusts the strategy mainly based on the feedback of rewards to achieve the learning goal.[12] In 2017, Spieker et al[11] applied RL in CI testing for the first time and proposed the frame of Reinforcement Test Case Selection Method (RETECS).

Figure 1 presents the integration of RL and CI testing.[11] The environment is the CI cycle, the state is the test cases submitted in each integration cycle, the action is the prioritization of the test cases, and the strategy is how to prioritize the test cases. The determination of the strategy is based on the interaction between the environment and the agent. The agent optimizes the next behavior of the state according to the feedback of historical behavior, which is measured with reward. So the quality of reward directly affects the quality of RL.

The RETECS[11] used the reward with current execution results to directly feedback the interaction between the agent and the environment. Only failure test cases are rewarded with the reward values in RETECS. In actual industry programs, with the low-failure test, RL only rewards a few failure test executions, resulting in sparse rewards. In addition, the reward based on the current execution result makes the failure test cases obtain the same reward value, and it is difficult for the agent to distinguish the priority among the failure test cases, which leads to the generalization of RL.[16]

The historical information-based reward[13] was proposed, including historical failure count and the average percentage of historical failure (*APHF*). Experiments have verified the effectiveness of the historical information-based reward in the RL-based CI testing. *APHF* can get a better TCP effect, which is defined as Formula 1.

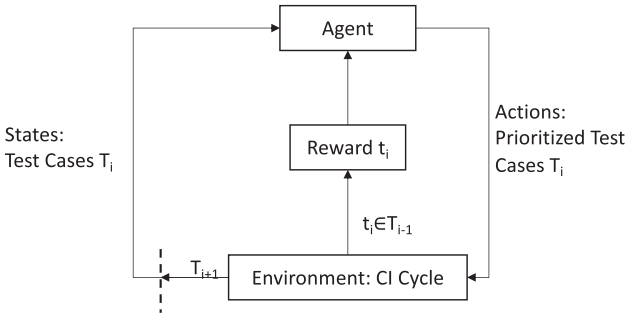

FIGURE 1 Reinforcement learning in continuous integration testing[11]

$$APHF_i(t) = \left(1 - \frac{\sum_{j=1}^{TH^{fail}} Rank_j}{TH^{fail} \times TH^{total}} + \frac{1}{2 \times TH^{total}}\right) \times t.verdict \quad (1)$$

here $Rank_j$ denotes the countdown order of the last jth failure of test case t, i is ith integration cycle of CI, TH^{fail} records the total failure number in historical execution of test case t, TH^{total} keeps track of the total execution number of test case t, and $t.verdict$ represents the current execution result of the test case t, with a value of 1 or 0, where 1 is failure execution and 0 is passed execution.

APHF uses the feature extraction of the historical failure distribution to achieve the measurement of test case fault detection capabilities. The value range is 0 to 1, which is a smaller value than the current execution result. Under ideal conditions, the best value of *APHF* is $\frac{1}{2}$ for continuous failure in history.

2.3 | Sparse reward of reinforcement learning

The sparse rewards of RL are common in real tasks, such as robotics[39] and autopilot.[40] The performance of the RL algorithm can be generally improved by solving the sparse reward problem. RL achieves the maximum expectation through the balance of exploration and utilization. The reward is the core of RL. The research to solve the sparse reward problem mainly includes reward design and learning,[15] experience playback mechanism,[41] exploration and utilization,[17] multi-objective learning,[42] and auxiliary tasks.[18] These methods are described below:

- The reward design method intuitively designs a 'dense' reward to solve sparse rewards.
- The experience replay mechanism method prioritizes the samples with higher priority in the experience library, which improves the utilization rate of the samples and is suitable for learning algorithms of departure strategy.
- The exploration and utilization method constructs virtual rewards to learn together with real rewards, which stimulates the learning process.
- The multi-objective learning method replaces the original target with a virtual target. Even at the beginning of training, the agent can quickly obtain rewards and greatly speed up the learning process at the same time.
- The auxiliary task method sets up auxiliary tasks to speed up learning and training.

In RL-based CI testing, the reward of the test case includes two aspects: the reward value and the reward object. According to the calculation formula in Formula 1, the reward value is affected by historical failure information. The reward object is determined by the current execution result with *t.verdict*. Most of the passed test cases cannot get effective rewards. The low-failure test is the main reason for sparse rewards of RL, where the reward value is too low and the reward objects are scarce.

In terms of reward value, *APHF* considers the distribution of historical failures and normalizes the reward value in [0, 1]. However, with full failure in history, the *APHF* value is only $\frac{1}{2}$. The more scattered the fault distribution, the lower the value of *APHF*. The small reward value will aggravate reward sparsity.

In terms of reward objects, only the failure test cases can be rewarded as reward objects. In actual CI testing, with low-failure test execution, most test executions are passed to be not rewarded as reward objects. The scarcity of reward objects leads to sparse rewards in RL.

3 | SPARSE REWARD FOR REINFORCEMENT LEARNING BASED TCP

Reward plays a vital role in RL.[12] However, with the high-frequency iteration and the low-failure test in the actual industrial CI testing, there is a sparse reward problem in RL-based CI testing. This paper solves the sparse reward problem from the perspective of increasing reward value and reward objects.

In this section, we define the historical failure density to objectively describe the sparse reward problem. Then, the reward based on the average failure position is proposed, where the influence of sparse reward is reduced by increasing the reward value. Finally, an additional reward method is proposed to increase the reward object. The test occurrence frequency is defined as an additional reward to be combined with any other rewards. The research framework for sparse rewards is presented in Figure 2.

3.1 | Historical failure density-based reward

In CI testing, the historical information of test cases is updated along with the integration process, which is a time sequence. The execution history of the test case t in the ith cycle can be defined as Test History(TH). TH records the execution information sequence of test case t in time series, with either 0 or 1. 0 means the passed execution and 1 means the failure execution. We use TH^{total}, TH^{fail}, and TH^{pass} to record the total historical execution number, the historical failure number, and the historical passed number, where $TH^{total} = TH^{fail} + TH^{pass}$. Since the test case does not occur in every cycle, i is not equal to TH^{total}.

In RL-based CI testing, the reward is obtained by extracting the feature of the test case to measure the test case's fault detection ability. Then, the reward will be fed back to the RL agent to realize the TCP strategy adjustment for the follow-up integration testing. The failure information can effectively evaluate the fault detection ability of the test case.[43] The evaluation of the test case can be directly used as a reward for the interaction between the environment and the agent. According to the reward study,[13] the historical failure information-based reward, especially the Average Percentage of Historical Failure (*APHF*), defined in Formula 1, can get a better TCP effect than the reward with current execution result (*TF*).

APHF is based on historical failure distribution, which takes into account the impact of historical failure execution on current failures. Theoretically, the denser the recent failures, the greater the *APHF* value. However, for all consecutive failures in history, the optimal value of *APHF* is $\frac{1}{2}$. For low-failure CI testing, the historical failure information of the test case is scarce, which correspondingly leads to a lower reward value. Therefore, in order to objectively measure the historical failure of test cases, the historical failure density(*HFD*) is first proposed. *HFD* extracts the characteristics of the test case's historical failure density, which can be used as a reward in Definition 1.

Definition 1. Historical Failure Density (HFD)-based Reward

$$HFD_i(t) = \frac{TH^{fail}}{TH^{total}} \times t.verdict \tag{2}$$

HFD objectively describes the sparseness of the test case's historical failures. For historical information with intensive failures, the denser the failures, the greater the *HFD* value. *HFD* can reach the optimal value of 1. For rare historical failure information, the *HFD* value will be correspondingly smaller.

Through the evaluation of *HFD*, the sparseness of historical failure information is visualized. For test cases with frequent historical failures, the value of *HFD* is close to 1. As the execution sequence increases, the impact of historical faults on the current integration gradually decreases. *HFD* is affected by the number of historical failures and historical executions.

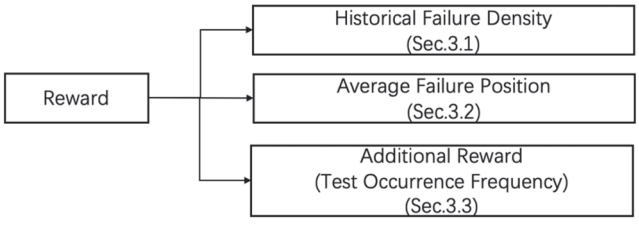

FIGURE 2 The reward research framework for sparse rewards

3.2 | Average failure position-based reward

HFD measures the historical failure density of test cases. For test cases with the same number of failures and executions, they will get the same *HFD* values. However, the different failure position will lead to different failure detection capabilities, where *HFD* cannot distinguish the reward values. For example, there are two test cases t_1 and t_2, and their historical execution records are $TH_{t_1} = [1,1,1,0,0,0]$ and $TH_{t_2} = [1,0,0,0,1,1]$, in reverse order. They all executed six times but failed three times. The value of *HFD* is equal to $\frac{1}{2}$ with the two test cases. However, t_1 has recently experienced three failures, and t_2 has recently experienced one failure and two early failures. In theory, t_1 has a better fault detection capability than t_2. Therefore, *HFD* cannot distinguish the difference among test cases that fail the same number of times in different cycles.

The historical failure distribution of the test case reflects the failures during the historical execution. *APHF* evaluates the fault detection ability of the test case based on historical failure distribution, but *APHF* ignores the sparse problem of historical failures. *HFD* can objectively measure the sparsity of historical failures, but *HFD* does not consider the distribution of historical failures. For the historical execution information with accidental failures, the reward value cannot be obtained due to the influence of the historical execution length. The reward value is too small to improve the impact of the sparse reward. There is a strong correlation between failure test and test failure. Therefore, the fault detection capability can be measured based on the failure location. The failure position is a relatively large value, which can effectively increase the reward value, thereby improving the sparse reward. In theory, the closer the failure location is to the current execution, the higher the fault detection capability of the test case. Therefore, we further propose a fault detection capability measurement method based on average failure position (*AFP*), which is defined in the Definition 2.

Definition 2. Average Failure Position (AFP)-based Reward

$$AFP_i(t) = \frac{\sum_{k=1}^{TH^{fail}} Rank_k}{TH^{fail}} \times t.verdict \tag{3}$$

here $Rank_k$ denotes the executive position of the last kth failure of the test case t.

AFP measures the average historical failure position based on the historical failure distribution, which can measure the failure detection capability of the test case. The closer the fault is to the current integration testing, the greater the value of *AFP* is. Early failures are gradually reduced with the influence of integrated algebra. Based on the *AFP* reward method, the reward values of t_1 and t_2 are 5 and 3, respectively. Their fault detection capabilities can be distinguished. The value of *AFP* is not only affected by the number of historical failures but also by the distribution of historical failures. Under the same number of failures, the closer the failure position is to the current integration testing, the greater the value of *AFP* will be. For the test case with accidental failure, *AFP* will get a huge reward value and can effectively increase its priority during testing.

3.3 | Test occurrence frequency-based additional reward

The fault detection capability of the test case can be effectively measured with the historical failure information. Through the reward design, the reward value of the test case is increased to realize the solution of the sparse reward in RL. However, in RL-based CI testing, only failure test cases are rewarded as reward objects. The low-failure test of CI testing makes only a small number of failure test cases obtain rewards, leading to scarce reward objects in RL.

Scarce reward objects lead to sparse rewards in RL-based CI testing. Test cases are submitted according to the test requirements. Once the test case fails, the fault in the CI system can be detected. Therefore, the test case has the failure effect, which determines the importance to the CI system. Test cases with failure effects should be rewarded as reward objects to increase their priority in the follow-up test sequence. At present, the failure effect of the test case only considers the current failure execution, where other information of the test case is ignored. A large amount of passed execution information indicates that most test cases cannot be effectively evaluated during their historical execution. RL does not reward most of the test execution, which intensifies the sparse reward. The reward design based on historical failure information improves the reward value of test cases with the frequent integration but cannot improve the scarcity of reward objects with the low-failure test of CI testing. Therefore, it is necessary to increase the reward objects to solve the sparse reward according to the failure effect of the test cases.

The failure effect of the test case determines whether the test case will be rewarded as a reward object.[13] Although the failure information of the test case can be used to measure the fault detection ability, the failure effect cannot be effectively evaluated for the passed execution. In

each integration cycle, test cases are submitted according to test requirements to ensure that there is no new faults introduced, thereby ensuring the normal operation of the CI system.[19] Therefore, some test cases have a strong correlation with the backbone, where the test cases are frequently submitted but rarely fail. These test cases have a large failure effect and are often tested during the integration process. Therefore, this paper further proposes test occurrence frequency(*TOF*) to measure the failure effect of test cases, which is defined in Definition 3.

Definition 3. Test Occurrence Frequency (TOF)

$$TOF_i(t) = \frac{TH^{total}}{i} \tag{4}$$

TOF measures the failure effect of the test case t from the overall perspective of the historical execution. The more frequently a test case is tested, the greater the *TOF* value will be. It should be noted that there may be multiple submissions of the test case for multiple modifications in one integration cycle. Therefore, the requirement-based test case may be submitted repeatedly, which causes the value of *TOF* greater than 1.

According to the failure effects, the test cases with large failure effects are further selected to be rewarded so as to solve the sparse reward problem caused by sparse reward objects. First, we recognize the failure effect of failure test cases on the integrated system. Second, the failure effect of the test case is analyzed to determine the identity of the reward object. Due to the low-failure test of CI testing, the historical information-based reward value is low. Therefore, through the analysis of *TOF*, the reward value of test cases with failure effect is added to solve the sparse reward problem. Therefore, this article proposes an additional reward method, which is defined as Definition 4.

Definition 4. Reward with Additional Reward

$$Reward\_Add_i(t) = \begin{cases} Reward_i(t) + TOF_i(t) & TOF_i(t) > \partial \\ Reward_i(t) & else \end{cases} \tag{5}$$

here *Reward* can be any reward used in RL-based CI testing, such as *TF* and historical failure information-based reward *APHF*, *HFD*, and *AFP*. ∂ is a threshold value used to judge the failure effect of a test case with *TOF*. That is, if the *TOF* exceeds a particular value, the test case is considered to be important, which will get an additional reward with *TOF*.

Based on the current execution, the failure test case as the reward object will get the reward value, which is calculated with the historical failure information. However, the passed test cases cannot be rewarded as reward objects in the original reward method. Under the reward with additional reward method, some passed test case can be rewarded as reward objects with the failure effect judgment of *TOF*, and the additional reward value based on *TOF* will be given to increase their priority in the follow-up test sequence. For an accidental failure test case, since there is less historical failure information, the reward value based on historical failure information is lower. The reward with additional reward method will be used to increase its priority by increasing the reward value with *TOF*. Therefore, the additional reward method, on the one hand, increases the reward for passed test cases and, on the other hand, increases the reward value of accidentally failure test cases, both of which can solve the sparse reward problem of RL.

Aiming at the sparse reward problem in RL-based CI testing, this paper proposes effective reward methods to increase the reward value or increase the reward objects. First, the historical failure density-based reward (*HFD*) and the average failure position-based reward (*AFP*) are defined to increase the reward value. In order to improve the reward objects, we propose an additional reward method based on the test occurrence frequency (*TOF*). The reward measures the failure effect of test cases from the overall perspective of the test execution history to determine the reward object. On the basis of rewarding failure test cases, parts of the passed test cases are added to be rewarded with the additional reward method.

4 | EXPERIMENTS

In this section, we present an experimental evaluation with different rewards. First, we propose relevant research questions based on the effectiveness of different rewards (Section 4.1). Then, the relevant experimental subjects (Section 4.2) are introduced. Before setting up the experiment (Section 4.4), we give the evaluation criteria systematically (Section 4.3). In Section 4.5, we present the experimental results and analysis. Section 4.6 is used to discuss and conclude the major results. Finally, Section 4.7 poses a threat to our work, thus ending the evaluation.

TABLE 1 Statistics of data sets

Data set	Test cases	CI cycles	Results	Failure rate	Frequency
Apache Commons	457	434 (28)	332,650	0.02%	1.68
Apache Drill	556	1,350 (26)	8,887	2.28%	0.01
Apache Hive	452	32 (4)	1,204	0.91%	0.08
Apache Parquet	273	1,409 (109)	205,770	0.15%	0.53
Apache Tajo	314	5,942 (994)	903,301	0.29%	0.48
DSpace	106	3,813 (218)	204,160	1.82%	0.51
Google Auto	46	638 (21)	14,601	0.19%	0.50
Google Closure	360	2,450 (89)	663,470	0.07%	0.75
Google Guava	433	807 (65)	877,466	0.02%	2.51
GSDTSR	5,555	336 (284)	1,260,618	0.25%	0.68
IOF/ROL	1,941	320 (271)	32,260	28.79%	0.05
Mybatis	303	988 (55)	815,598	0.15%	2.72
Paint Control	89	352 (257)	25,594	19.36%	0.82
Rails	2,010	3,263 (2,047)	781,273	0.62%	0.12

4.1 | Research questions

The experiments and analyses are based on the following research questions.

- How effective are the historical failure information-based rewards of *HFD* and *AFP* to reduce the impact of sparse rewards in TCP of RL-based CI testing?
- How does the additional reward method solve the sparse reward problem effectively?
- How does the reward with additional reward affect the efficiency of the RL-based CI testing framework?

All questions are set up to verify the rewards to solve the sparse reward problem with the TCP effect in RL-based CI testing. **RQ1** is used to demonstrate the improvement of the sparse reward through the TCP effect with the historical failure information-based rewards (*HFD* and *AFP*). **RQ2** is set up to study the effectiveness of the reward with additional reward in solving the sparse rewards, compared to the reward without additional reward. **RQ3** is used to study the efficiency of the reward with additional reward in the RL-based CI testing framework.

4.2 | Experimental subjects

Fourteen data sets were used in the experiments, which were shared by other researchers for industrial research in related fields. We collected and processed the data sets to make them suitable for RETECS. Paint Control and IOF/ROL are from ABB Robotics Norway for testing sophisticated industrial robots.* GSDTST[†] and Rails[‡] are open-source data sets shared by researchers. The other 10 data sets are shared online.[§] The continuous integration test log files are analyzed in real time to obtain the relevant information of test cases during each test execution, including the ID of the test case, test history, test result, running time, and integration cycle. Some of the processed data are provided on the website[‖] by Spieker et al.[11] The test case is a unique identifier, which will be used to test different commits in different cycles. There are test execution results of each integration cycle in these data sets.

Table 1 lists the details of the fourteen data sets, including the number of test cases, the number of CI cycles, the number of execution results, failure rate, and frequency. CI cycles indicate the number of CI cycles and the actual number of failure cycles as noted later. Results is the number of execution results, which refers to the total number of all test cases' executions during the whole CI process. Failure rate represents the failure ratio of total executions, and Frequency is the probability of each test case occurring in each cycle. There were six data sets with CI cycles greater than 1,000, three cycles between 500 and 1,000, and five integration cycles below 500. Among the 14 data sets, 2 data sets have a failure rate of more than 19%, which is a high failure rate, and two 2 sets have a failure rate between 1% and 3%, while the other 10 data sets have a failure rate less than 1%, which is very consistent with the actual industrial situation. Frequency emphasizes the frequency of each test case in each cycle. There are only three data sets whose frequency is greater than 1, and for most data sets, their frequency is less than 1. On the basis

of rewarding failure test cases, the low failure and low frequency of CI testing will lead to sparse rewards in RL. TCP in CI testing is complicated when test cases occur infrequently or fail inefficiently.

4.3 | Evaluation measures

To effectively conduct experimental verification, we analyze the TCP effect of Rl-based CI testing with four evaluation measures, including NAPFD (Normalized Average Percentage of Faults Detected), Recall, TTF (Test To Fail), and Time Consumption.

NAPFD[44] is adopted as the evaluation metric, which is defined as follows:

$$NAPFD(TS_i) = p - \frac{\sum_{j \epsilon TS_i^{fail}} rank(j)}{\left|TS_i^{fail}\right| \times |TS_i|} + \frac{p}{2 \times |TS_i|}. \quad (6)$$

With $p = \frac{|TS_i^{fail}|}{|TS_i^{total,fail}|}$, $rank(j)$ represents the position of the jth failure test case in the test sequence TS_i, $\left|TS_i^{fail}\right|$ indicates the total number of test cases failed in TS_i, $|TS_i|$ indicates the total number of test cases in TS_i, and $\left|TS_i^{total,fail}\right|$ indicates the total number of test cases failed in the TS.

Rothermel et al[45] proposed Average Percentage of Faults Detected(APFD) to evaluate the effectiveness of TCP techniques. APFD evaluates a test sequence based on the index of the failure test cases in the test sequence. However, when combining TCP with test case selection, not all test cases are executed, so not all faults can be detected. But APFD assumes that all defects are detected. APFD only applies to situations where there is no test case selection. NAPFD[44] is an extension of APFD, which reflects the proportion of faults detected to all faults, and is suitable for the existence of test case selection. If all faults are detected, NAPFD is the same as APFD ($p = 1$). When calculating the NAPFD value in this experiment, we assume that the fault corresponding to each failure is different.

Recall is defined as the percentage of total faults found. Only test cases with half of the total execution time are selected for execution, which can simulate the rapid feedback mechanism of the CI system. Some test cases could detect failures not being implemented. TCP needs to prioritize test cases that may fail as much as possible. Therefore, Recall can be used to evaluate the ordering effect of TCP.

TTF (Test to Fail) is another evaluation index, which measures the first failure position in the test sequence. The more advanced the first failure position, the more timely feedback. So the smaller the TTF value is, the higher the first failure position is found.

Time consumption measures the average time overhead in the RL-based CI testing framework, including the time of the reward calculation, test case prioritization, and agent learning.

In general, four evaluation metrics, NAPFD, Recall, TTF, and Time Consumption, are used to compare the TCP effect of different rewards under the RL-based CI testing framework, to verify the improvement of the sparse reward problem.

4.4 | Experimental setup

The experiments conducted in this paper are based on the framework of RETECS.[#] There are eight rewards of RL implemented, including *TF*,[11] *APFH*,[13,14] *HFD*, *AFP*, and the four rewards with *TOF*-based additional reward. *TF*[11] and *APFH*[13] are selected as baseline, where *TF* is based on the current test case failure and *APFH*[13] is based on historical failure distribution. Furthermore, *TF*, *APHF*, *HFD*, and *AFP* are conducted with the additional reward to verify the effectiveness of *TOF*-based additional reward.

In the experiments, the parameters are consistent with RETECS, which is proved optimal by Spieker et al.[11] For the agent selection, the network-based agent is selected, which is better for CI testing in RETECS. The network-based agent is a multilayer perceptron neural network algorithm. However, the combination of RL and neural network may lead to instability of the algorithm.[12] There are 60 repeated experiments in this pare to eliminate the influence of random factors.

Due to the strong correlation between failure test and test failure, the time window method is adopted to select the recent historical information. The time window based-reward is proved better than the whole historical information-based reward.[13] The time window size of five is selected according to the research of Wu et al.[14] The historical failure information-based reward is calculated based on the time window. On this basis, the time window-based reward with additional reward is conducted to compare the TCP effect.

For the reward with additional reward, the failure effect of test cases is judged based on the *TOF* threshold. Therefore, in order to select the appropriate threshold value, we constructed the tunning experiment, where the parameters are adjusted from (0.5, 1, 1.5, 2). For these 14 data sets, the parameters of *TOF* for different rewards (*TF*, *APHF*, *HFD*, *AFP*) are (0.5, 2, 0.5, 1), (0.5, 0.5, 0.5, 0.5), (0.5, 0.5, 0.5, 0.5), (1.5, 1, 2, 2), (0.5, 1.5, 1, 1.5), (1.5, 1, 1, 1), (1.5, 1, 0.5, 0.5), (1.5, 1.5, 2, 1), (2, 2, 2, 2), (1, 2, 1.5, 2), (1, 2, 1.5, 0.5), (2, 2, 1.5, 2), (1.5, 0.5, 0.5, 2), and (1, 1.5, 1, 1) to add additional reward, respectively.

TABLE 2 The average TCP effect (standard deviation) of different rewards

	NAPFD (%)				Recall (%)				TTF			
Data set	TF	APHF	HFD	AFP	TF	APHF	HFD	AFP	TF	APHF	HFD	AFP
Apache Commons	21.45 (.004)	21.40 (.007)	21.52 (.005)	21.45 (.004)	100 (0)	100 (0)	100 (0)	100 (0)	409.94 (2.84)	410.57 (3.28)	409.42 (3.77)	410.00 (2.86)
Apache Drill	35.02 (.022)	35.57 (.026)	36.75 (.037)	35.56 (.016)	62.53 (.026)	59.42 (.040)	62.51 (.019)	61.75 (.024)	8.70 (1.39)	10.95 (.703)	8.95 (2.26)	8.94 (1.19)
Apache Hive	56.54 (.090)	59.60 (.103)	55.79 (.116)	70.83 (.060)	43.48 (.228)	49.13 (.255)	46.80 (.258)	48.37 (.068)	143.34 (83.1)	154.96 (80.6)	135.09 (80.0)	188.68 (32.9)
Apache Parquet	36.31 (.003)	36.84 (.008)	37.51 (.005)	36.93 (.007)	91.57 (.000)	91.14 (.025)	91.69 (.003)	91.55 (.006)	67.81 (.416)	66.35 (3.29)	66.26 (.661)	67.43 (.806)
Apache Tajo	17.55 (.012)	23.76 (.009)	24.56 (.018)	24.55 (.017)	74.41 (.077)	73.07 (.103)	80.23 (.011)	75.08 (.011)	116.49 (18.6)	99.88 (22.2)	113.20 (3.76)	101.12 (17.9)
DSpace	34.27 (.004)	37.21 (.010)	35.34 (.005)	36.76 (.008)	77.94 (.008)	78.80 (.006)	78.11 (.010)	78.46 (.008)	17.84 (.147)	13.23 (.807)	17.26 (.172)	13.20 (.438)
Google Auto	28.72 (.000)	29.83 (.011)	29.86 (.005)	28.54 (.000)	59.52 (0)	59.60 (.006)	59.52 (0)	59.52 (0)	10.19 (.000)	9.69 (.434)	9.64 (371)	10.29 (.000)
Google Closure	13.17 (.002)	13.68 (.0001)	13.84 (.009)	13.88 (.007)	61.25 (.002)	61.21 (.000)	61.33 (.006)	61.42 (.007)	135.83 (.050)	135.82 (.050)	135.70 (.639)	135.84 (.070)
Google Guava	23.50 (.063)	23.51 (.058)	19.30 (.010)	27.54 (.080)	91.68 (.025)	91.76 (.023)	89.39 (.066)	93.86 (.035)	741.59 (50.6)	742.73 (45.4)	766.77 (63.1)	712.86 (63.0)
GSDTSR	13.80 (.028)	49.85 (.162)	13.59 (.018)	42.10 (.187)	18.10 (.031)	54.59 (.162)	18.06 (.023)	46.82 (.189)	492.47 (86.2)	212.73 (114)	520.43 (27.5)	244.21 (127)
IOF/ROL	25.22 (.037)	41.77 (.006)	22.91 (.019)	32.08 (.056)	34.71 (.042)	51.37 (.007)	31.96 (.022)	41.42 (.058)	6.62 (1.91)	1.34 (.109)	7.87 (11.57)	4.47 (2.66)
Mybatis	27.97(.011)	29.88(.023)	29.12(.021)	29.86(.021)	65.35(.018)	65.26(.021)	65.31(.021)	65.34(.018)	300.59(3.47)	281.41(15.5)	288.82(14.4)	281.88(15.8)
Paint Control	65.15 (.029)	66.73 (.022)	23.41 (.079)	64.71 (.045)	76.18 (.030)	78.36 (.022)	32.83 (.094)	76.99 (.033)	3.74 (.349)	3.82 (.594)	8.12 (.500)	4.33 (1.25)
Rails	62.63 (.102)	87.10 (.011)	61.92 (.183)	87.84 (.009)	68.81 (.091)	90.89 (.011)	70.37 (.155)	91.69 (.007)	13.63 (4.36)	4.41 (.566)	17.68 (11.2)	4.07 (.580)

Before the TCP evaluation, test case selection is used to simulate the rapid feedback mechanism of CI testing. The test case selection mechanism selects test cases, which run until half of the total execution time in the cycle.

4.5 | Experimental results

In this section, the experimental results are presented to answer each research question, respectively.

4.5.1 | Analysis of RQ1

To answer RQ1, we compare different rewards with TCP effects to verify the improvement of sparse rewards. *TF* is the best reward with current execution results.[11] *APHF* is the best historical information-based reward in previous paper.[13] We choose *TF* and *APHF* as benchmarks for comparison. The two historical failure information-based rewards *HFD* and *AFP* compare the TCP effect with *TF* and *APHF* in RL-based CI testing. The historical failure information is conducted under the time window size of 5, which is proved better in the time window-based reward method.[14] The TCP effect is compared with NAPFD, Recall, and TTF. Table 2 describes the average values of the three indicators (NAPFD, Recall, and TTF) on different data sets in the columns of NAPFD, Recall, and TTF with standard deviation marked. The gray background is used to highlight the optimal value based on one indicator.

For the NAPFD comparison in the columns of NAPFD in Table 2, on the overall average, the average NAPFD ordering effect of different rewards is *APHF* > *AFP* > *TF* > *HFD*. However, with the highlighted background, there are five data sets best at *APHF*, five data sets best at *HFD*, and four data sets best at *AFP*. With the comparison of *HFD* and *TF*, although *HFD* is weaker than *TF* on average NAPFD on the whole, 10 data sets perform better on average NAPFD with *HFD* than that with *TF*. With the comparison of *AFP* and *TF*, except for Google Auto, 13 data sets perform better in *AFP*. Among different historical failure information-based rewards, compared with *APHF*, *HFD* is better in six data sets, and *AFP* is better in seven data sets. The low frequency makes the situation of the TCP complex.

With the comparison of average Recall in the columns of Recall in Table 2, on the overall average, the ranking of the rewards in Recall is *APHF* > *AFP* > *TF* > *HFD*. However, except for Apache Commons, with the highlighted background, there are two data sets best at *TF*, six data sets best at *APHF*, two data sets best at *HFD*, and three data sets best at *AFP*. With the comparison of *HFD* and *TF*, there are six data sets better in *HFD* and two data sets with the same Recall. With the comparison of *AFP* and *TF*, there are 10 data sets better in *AFP* and two data sets with the same Recall. With the comparison of *HFD* and *APFH*, there are five data sets better in *HFD* and one data set with the same Recall. With the comparison of *AFP* and *APHF*, there are seven data sets better in *AFP* and one data set with the same Recall.

With the comparison of average TTF in the columns of TTF in Table 2, on the overall average, the ranking of TTF in different rewards is *APHF* > *AFP* > *TF* > *HFD*. However, with the highlighted background, there are two data sets best at *TF*, four data sets best at *APHF*, five data sets best at *HFD*, and three data sets best at *AFP*. With the comparison of *HFD* and *TF*, there are eight data sets better in *HFD*. With the comparison of *AFP* and *TF*, there are eight data sets better in *AFP*. With the comparison of *HFD* and *APFH*, there are six data sets better in *HFD*. With the comparison of *AFP* and *APHF*, there are five data sets better in *AFP*.

Comparing NAPFD, Recall, and TTF, different characteristics of the data sets lead to different TCP effects with different rewards. That is, the most suitable reward for each data set is different. With the comparison of NAPFD, compared with *TF*, in most data sets, *HFD* and *AFP* can effectively improve the sparse reward, except for Paint Control, which is a high failure data set. With the comparison of NAPFD, compared with *APHF*, there are nine data sets improved in sparse reward with *HFD* or *AFP*.

On the basis of the average value, standard deviation analysis is further carried out for the evaluation results, as shown in the marked of Table 2. Standard deviation measures the dispersion of the average value. Since TTF measures the first failure location, the randomness is large, and it is difficult to measure the convergence. Except for TTF, the standard deviations of NAPFD and Recall are very small, which means that the value obtained each time is very close to the mean value. On some data sets, the convergence is improved by increasing the reward value to improve the sparse rewards, such as Rails in *AFP*.

Further, Tukey Honest Significant Difference(Tukey HSD) is proposed to compare the significance of the TCP difference between the two reward functions. Tukey HSD is the multiple comparisons of means, which is based on the Analysis of Variance(ANOVA) in Appendix 3. The ANOVA tests the significant difference between the two sample means, which is based on *F* and *P* values. In Appendix A.1, the ANOVA proves that in most data sets, through the solutions of sparse rewards, the TCP metrics are significantly different. The results of Tukey HSD are listed in Table 3. SR_1 is Tukey HSD result compared to TF, and SR_2 is Tukey HSD result compared to APHF, with T for true and F for false. + and − indicate the negative and positive of T and F. For example, *T*(−) indicates that the data are significantly worse than the original data, *T*(+) indicates that the data are significantly better than the original data, *F*(+) indicates that the data are not significantly worse than the original data, and *F*(−) indicates that the data are better than the original data. Therefore, *T*(−) is the rejected result.

With the *T*(−) results in Table 3, there are 28.57% rejection results in total. In the columns of NAPFD, the number of *T*(−) is (4, 6, 1, 3). In the columns of Recall, the number of *T*(−) is (2, 6, 0, 5). And in the columns of TTF, the number of *T*(−) is (5, 8, 3, 5). According to the statistical results, the Tukey HSD comparison with TF is stronger than the Tukey HSD comparison with APHF. With the failure rate increased, such as Paint control, the two new historical information-based rewards (*HFD* and *AFP*) are difficult to adapt to the test environment. In most cases, *HFD* and *AFP* can produce a certain positive difference value, with the TCP effect improved.

However, historical failure information-based rewards are significantly better than the reward based on current execution. The two new historical information-based rewards (*HFD* and *AFP*) proposed in this paper can reduce the sparse reward problem by increasing the reward value in a certain process, thereby significantly improving the TCP effect.

4.5.2 | Analysis of RQ2

To answer RQ2, the TCP effect is compared with the rewards without additional reward and the reward with additional reward in Tables 2 and 4. Table 2 presents the TCP effect of the rewards without additional reward, and Table 4 presents the TCP effect of the reward with additional reward. The TCP effect is compared with NAPFD, Recall, and TTF, which are shown in the columns of NAPFD, Recall, and TTF in Table 2 and Table 4 with standard deviation marked. The columns describe the three indicators (NAPFD, Recall, and TTF) on different data sets. With the comparison of Table 2 and Table 4, a better index value of the reward with additional reward is highlighted with gray background in Table 4. It means the reward with additional reward can get a better TCP effect than the reward without additional reward.

Figure 3 further compares the difference of NAPFD in the cycle of the rewards with or without additional reward. The *x* coordinate is the integration cycle, and the *y* coordinate is the NAPFD value. If there is a positive value, it indicates that the reward with additional reward performs better in the current cycle, while the reward without additional reward performs better on the contrary.

We compare the three indicators in Tables 2 and 4. There are 69.05% background highlighted with gray in Table 4. In most cases, compared with the rewards without additional reward in Table 2, the reward with additional reward can improve the TCP performance more effectively. We further compare three different indicators separately.

The average NAPFD is compared in the columns of NAPFD in Tables 2 and 4. On the overall average, for each reward, the average NAPFD of the reward with the additional reward can be improved. There are 13 data sets enhanced with *TF_Add* compared to *TF*, 10 data sets improved with *APHF_Add* compared to *APHF*, 11 data sets improved with *HFD_Add* compared to *HFD*, and 8 data sets improved with *AFP_Add* compared to

TABLE 3 Turkey HSD results of different rewards comapred to TF and APHF

	NAPFD				Recall				TTF			
	HFD		AFP		HFD		AFP		HFD		AFP	
Data set	HSD_1	HSD_2	HSD_1	HSD_2	HSD_1	HSD_2	HSD_1	HSD_2	HSD_1	HSD_2	HSD_1	HSD_2
Apache Commons	F(−)	F(−)	F(+)	F(−)	F(+)	F(+)	F(+)	F(+)	F(−)	F(−)	F(+)	F(−)
Apache Drill	T(+)	T(+)	F(−)	F(+)	F(+)	T(+)	F(+)	T(+)	F(+)	T(+)	F(+)	T(+)
Apache Hive	F(+)	F(−)	T(+)	T(+)	F(−)	F(+)	T(+)	T(−)	F(−)	F(−)	T(−)	T(−)
Apache Parquet	T(-)	T(+)	T(+)	F(+)	T(+)	F(−)	F(−)	F(+)	T(+)	F(−)	T(+)	T(−)
Apache Tajo	T(+)	T(+)	T(+)	T(+)	T(+)	T(+)	F(−)	F(−)	F(−)	T(−)	T(+)	F(+)
DSpace	T(+)	T(−)	T(+)	T(−)	F(−)	T(−)	T(+)	T(−)	T(+)	T(−)	T(+)	F(−)
Google Auto	T(+)	F(−)	T(−)	T(−)	F(+)	F(+)	F(+)	F(+)	T(+)	T(+)	T(−)	T(−)
Google Closure	F(−)	F(−)	F(v)	T(+)	F(−)	F(−)	F(−)	T(+)	F(−)	F(−)	F(+)	F(+)
Google Guava	T(−)	T(−)	T(+)	T(+)	T(−)	T(−)	T(+)	T(+)	T(−)	T(−)	T(+)	T(+)
GSDTSR	F(+)	T(−)	T(+)	T(−)	F(+)	T(−)	T(+)	T(−)	T(−)	T(−)	T(+)	F(+)
IOF/ROL	T(−)	T(−)	T(+)	T(−)	T(−)	T(−)	T(+)	T(−)	T(−)	T(−)	T(+)	T(−)
Mybatis	T(+)	F(+)	T(+)	F(+)	F(+)	F(−)	F(+)	F(+)	T(+)	T(−)	T(+)	F(+)
Paint Control	T(−)	T(−)	F(+)	T(−)	T(−)	T(−)	F(−)	T(−)	T(−)	T(−)	T(−)	T(−)
Rails	F(+)	T(−)	T(+)	T(+)	F(−)	T(−)	T(+)	T(+)	T(−)	T(−)	T(+)	T(+)

Note: T is the True with significant difference, and F is the False without significant difference. HSD_1 is the significant result compared to *TF*, and HSD_2 is the significant result compared to *APHF*.

TABLE 4 The average TCP effect of different rewards with additional reward

	NAPFD (%)				Recall (%)				TTF			
Data set	TF_Add	APHF_Add	HFD_Add	AFP_Add	TF_Add	APHF_Add	HFD_Add	AFP_Add	TF_Add	APHF_Add	HFD_Add	AFP_Add
Apache Commons	21.80 (.023)	21.31 (.023)	21.73 (.010)	21.92 (.030)	91.61 (.188)	96.57 (.126)	99.76 (.009)	92.81 (.185)	364.53 (95.5)	395.50 (57.5)	406.89 (10.5)	373.31 (87.4)
Apache Drill	51.64 (.071)	42.83 (.040)	53.55 (.057)	41.72 (.027)	74.75 (.060)	67.16 (.057)	76.07 (.067)	67.56 (.037)	7.83 (1.30)	10.24 (1.13)	7.77 (1.07)	8.92 (1.32)
Apache Hive	43.48 (.238)	49.13 (.279)	46.80 (.222)	48.37 (.199)	52.38 (.272)	70.63 (.229)	60.46 (.200)	66.46 (.134)	45.89 (55.7)	100.21 (86.7)	54.05 (79.1)	90.43 (83.3)
Apache Parquet	36.52 (.003)	36.86 (.006)	37.48 (.006)	37.06 (.006)	91.57 (.003)	91.63 (.002)	91.57 (.003)	91.25 (.007)	67.58 (.167)	67.14 (1.03)	66.00 (.470)	66.63 (1.06)
Apache Tajo	18.75(.006)	24.28(.001)	24.91(.020)	24.53(.012)	78.80(.002)	76.12(.011)	80.17(.011)	74.32(.011)	125.83(1.63)	104.54(13.0)	112.29(4.41)	100.11(20.7)
DSpace	34.40 (.001)	37.73 (.015)	35.39 (.005)	37.30 (.009)	78.25 (.000)	78.78 (.005)	78.12 (.004)	78.92 (.003)	17.86 (.013)	12.99 (.804)	17.24 (.132)	12.92 (.335)
Google Auto	28.89 (.009)	29.79 (.009)	30.17 (.005)	28.59 (.001)	59.76 (.010)	59.52 (0)	59.52 (0)	59.52 (0)	10.21 (.037)	9.68 (.437)	9.49 (.257)	10.26 (.042)
Google Closure	13.99 (.010)	14.08 (.012)	13.84 (.009)	14.01 (.009)	61.53 (.010)	61.63 (.013)	61.34 (.007)	61.55 (.009)	135.82 (.068)	135.84 (.097)	135.74 (.477)	135.83 (.055)
Google Guava	25.94 (.081)	28.08 (.080)	19.61 (.012)	26.11 (.078)	90.35 (.135)	93.79 (.036)	89.64 (.043)	93.02 (.035)	692.58 (113)	706.76 (62.5)	767.39 (51.8)	721.44 (60.4)
GSDTSR	23.42 (.120)	27.90 (.128)	18.58 (.147)	28.42 (.137)	28.29 (.127)	32.57 (.131)	22.89 (.154)	33.05 (.141)	339.21 (132)	367.05 (103)	446.40 (137)	353.98 (99.7)
IOF/ROL	28.87 (.051)	41.94 (.006)	24.46 (.030)	27.93 (.050)	38.68 (.056)	51.62 (.009)	33.78 (.035)	37.46 (.052)	5.39 (2.24)	1.42 (.162)	6.79 (1.78)	5.00 (1.92)
Mybatis	31.30 (.022)	31.43 (.012)	30.45 (.026)	31.58 (.006)	65.18 (.030)	65.13 (.035)	61.49 (.109)	65.53 (.005)	294.10 (9.34)	264.47 (29.7)	244.27 (69.0)	266.85 (6.02)
Paint Control	65.79 (.012)	67.30 (.017)	41.07 (.105)	66.65 (.018)	76.86 (.014)	78.76 (.015)	54.70 (.107)	78.42 (.013)	3.65 (.240)	3.62 (.301)	8.51 (1.96)	3.85 (.746)
Rails	64.18 (.101)	87.21 (.007)	76.79 (.075)	87.69 (.008)	69.85 (.090)	90.95 (.007)	81.99 (.071)	91.59 (.006)	12.46 (4.45)	4.39 (.485)	8.33 (2.57)	4.29 (1.06)

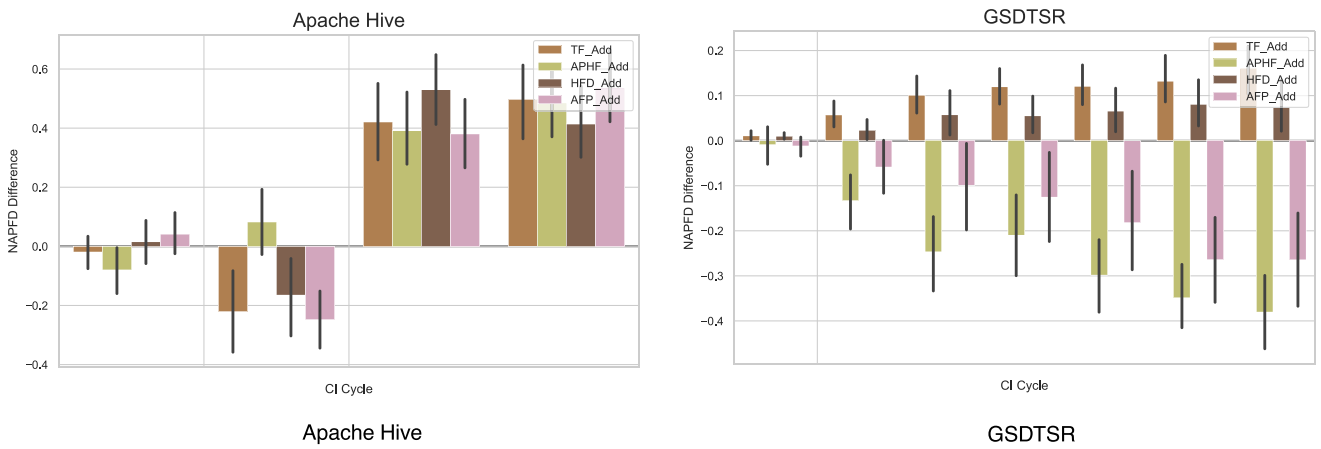

FIGURE 3 The NAPFD difference in cycle between the reward and the reward with additional reward

AFP. For Apache Hive, the reward with additional reward cannot enhance the TCP effect with average NAPFD. With further analysis of the execution history of the test cases in Apache Hive, the historical failure density of test cases is only 0 and 1, where the average failure rate from the overall execution is 0.91%.

The NAPFD difference is further compared with the reward with or without additional reward in the cycle of Apache Hive and GSDTSR in Figure 3. The NAPFD values of them do not get improved significantly in Table 4. With the NAPFD difference in Apache Hive, while there is no significant improvement in the global mean in Table 4, the NAPFD value is significantly improved with the continuous integration process, especially in the later stages of integration. For GSDTSR, the additional reward is not effectively suitable for *APHF* and *AFP* on each cycle, which presents negative values of *APHF_Add* and *AFP_Add* in Figure 3, consistent with the overall results of Table 4.

With the comparison of Recall in the columns of Recall in Table 2 and Table 4, the reward with additional reward can improve the Recall value of some rewards, especially *TF* and *HFD*. With the comparison of the reward with additional reward and the reward without additional reward, there are 10 data sets improved in *TF_Add*, 9 data sets improved in *APHF_Add*, 9 data sets improved in *HFD_Add*, and 6 data sets improved in *AFP_Add*. Therefore, for most programs, the reward with additional reward can effectively improve Recall.

With the comparison of TTF in the columns of TTF in Tables 2 and 4, on the overall average, except for *APHF*, the reward with addition reward can effectively improve the first failure position, where there is a smaller TTF value. With the comparison of the reward with additional reward and the reward without additional reward, there are 10 data sets improved in *TF_Add*, 9 data sets improved in *APHF_Add*, 9 data sets improved in *HFD_Add*, and 10 data sets improved in *AFP_Add*. Therefore, for most programs, the reward with additional reward can effectively improve TTF.

On the basis of the average value, standard deviation analysis is further carried out for the evaluation results, as shown in the marked of Table 4. Standard deviation measures the dispersion from the specific value to the average value. The first failure location(TTF) fluctuates greatly, resulting in very discrete results. So the standard deviation of TTF is relatively bigger. Except for TTF, the standard deviations of NAPFD and Recall are very small, which means that the specific value is very close to the average value. The reward with additional reward not only improves the TCP learning effect of RL but also improves the degree of convergence to a certain extent.

Further, Tukey HSD is proposed to compare the significant difference of the reward with additional reward and the reward without additional reward. Tukey HSD is the multiple comparisons of means, which is based on the Analysis of Variance (ANOVA) in Appendix A.2. The ANOVA proves that in most data sets, through the solution of sparse rewards, the TCP metrics of different rewards are significantly different. The results of Tukey HSD are shown in Table 5. T is True for significant difference, and F is False for insignificant difference with + and − marked. $T(-)$ is the rejected result, which means the index value of the reward with additional reward is far worse than the index value of the reward without additional reward.

With the $T(-)$ results in Table 5, there are only 13.69% rejection results in total. In the columns of NAPFD, the number of $T(-)$ is (1,2,1,3). In the columns of Recall, the number of $T(-)$ is (1,2,3,4). And in the columns of TTF, the number of $T(-)$ is (2,2,0,3). According to the statistical results, the reward with additional reward can produce a better TCP effect than the reward without additional reward, where the number of $T(-)$ is small. Therefore, the reward with additional reward can optimize the TCP effect in most cases.

In general, by comparing the TCP effect of the reward with and without additional reward on three metrics(NAPFD, Recall, and TTF), the reward with additional reward can effectively improve their TCP performance in most data sets. There are 10 data sets with the best NAPFD value produced by the reward with additional reward, 8 data sets with the best Recall value produced by the reward with additional reward, and

TABLE 5 Turkey HSD results of different rewards with additional reward compared to the rewards without additional reward

	NAPFD				Recall				TTF			
Data set	TF_Add	APHF_Add	HFD_Add	AFP_Add	TF_Add	APHF_Add	HFD_Add	AFP_Add	TF_Add	APHF_Add	HFD_Add	AFP_Add
Apache Commons	F(−)	F(+)	F(−)	F(−)	T(−)	T(−)	T(−)	T(−)	T(+)	T(+)	F(−)	T(−)
Apache Drill	T(+)	T(+)	T(+)	T(+)	T(+)	T(+)	T(+)	T(+)	T(+)	T(+)	T(+)	F(−)
Apache Hive	T(-)	T(-)	T(-)	T(-)	F(-)	T(+)	F(-)	T(+)	T(+)	T(+)	T(+)	T(+)
Apache Parquet	T(+)	F(−)	F(+)	F(−)	F(+)	F(−)	T(−)	T(−)	T(+)	F(+)	T(+)	T(+)
Apache Tajo	T(+)	T(+)	F(−)	F(+)	T(+)	F(−)	F(+)	F(+)	T(−)	F(+)	F(−)	F(−)
DSpace	F(−)	T(+)	F(−)	T(+)	T(+)	F(+)	F(−)	T(+)	F(+)	F(−)	F(−)	T(+)
Google Auto	F(−)	F(+)	T(+)	T(+)	F(−)	F(+)	F(+)	F(+)	T(−)	F(−)	T(+)	T(+)
Google Closure	T(+)	T(+)	F(+)	F(−)	T(+)	T(+)	F(−)	F(−)	F(−)	F(+)	F(+)	F(−)
Google Guava	F(−)	T(+)	F(−)	F(+)	F(+)	T(+)	F(-)	F(+)	T(+)	T(+)	F(+)	F(+)
GSDTSR	T(+)	T(−)	T(+)	T(−)	T(+)	T(−)	T(+)	T(−)	T(+)	T(−)	T(+)	T(−)
IOF/ROL	T(+)	F(−)	T(+)	T(−)	T(+)	F(−)	T(+)	T(−)	T(+)	T(−)	T(+)	F(+)
Mybatis	T(+)	T(+)	T(+)	T(+)	F(+)	F(+)	T(−)	F(−)	T(+)	T(+)	T(+)	T(+)
Paint Control	F(−)	F(-)	T(+)	T(+)	F(-)	F(-)	T(+)	T(+)	F(-)	T(+)	F(+)	T(-)
Rails	F(−)	F(−)	T(+)	F(+)	F(−)	F(−)	T(+)	F(+)	F(−)	F(−)	T(+)	F(+)

T is the True with significant difference, and F is the False without significant difference.

9 data sets with the best TTF value produced by the reward with additional reward. A better TCP performance means that the reward with additional reward is more helpful to know RL for CI testing. Through the standard deviation and ANOVA analyses, the improvement of the TCP effect is not a random result. Therefore, the reward with additional reward effectively solves the sparse reward problem with the increases of reward objects and reward values.

4.5.3 | Analysis of RQ3

To answer RQ3, the comparison of the Time Consumption is conducted for all four rewards with and without additional reward. Table 6 lists the average Time Consumption for all 14 data sets. The reward with additional reward will increase the time cost by taking extra information into account. The statistics of Time Consumption are based on the RL-based CI testing framework, which involves the time cost of reward calculation, agent learning, and test case prioritization. The reward for RETECS is to get a better test case ranking effect and get better learning efficiency, which accelerates the convergence of RL. With the additional reward, the Time Consumption increases, such as Apache Commons, with a maximum growth of 0.15s. However, in some data sets, such as GSDTSR, the Time Consumption of the reward with additional reward is reduced, which accelerates the convergence of RL.

In general, through the comparison of Time Consumption in Table 6, the reward with additional reward increases the Time Consumption within the acceptable range of seconds. Based on solving the sparse reward problem effectively, the reward with additional reward has little impact on the efficiency of the framework.

4.6 | Discussion

In CI testing, there are frequent iteration and quick feedback in need. RL-based CI testing, through the balance of exploration and utilization, generates a TCP strategy suitable for follow-up integration testing. Reward, as the core of RL, guides the agent to adjust the TCP strategy. However, in actual industrial programs, the high-frequency iteration and the low-failure test of CI testing lead to the sparse reward of RL. The research on the reward is carried out with different reward methods designed to solve the sparse reward problem.

We respectively discuss the three reward methods proposed in this article.

- *HFD* is designed based on the historical failure density of the test case, which objectively describes the historical failure sparseness of the test case. Compared to *TF* and *APHF*, there is a significant difference in the TCP effect of *HFD* in the column of P_1 in Table C2. Therefore, the historical failure sparseness of the test case has a significant impact on the TCP effect. With the historical information-based reward method (*APHF* and *HFD*), for the data sets with high failure rates such as Paint Control and IOF/ROL, *HFD* reduces the average TCP effect significantly, which is proved with Turkey HSD in Table 3. In the continuous integration process, the historical failure sparsity is constantly changing, usually

TABLE 6 The average time consumption of different rewards(s)

Data set	TF	APHF	HFD	AFP	TF_Add	APHF_Add	HFD_Add	AFP_Add
Apache Commons	0.08	0.08	0.22	0.22	0.23	0.23	0.23	0.23
Apache Drill	0.003	0.008	0.004	0.005	0.005	0.01	0.005	0.005
Apache Hive	0.04	0.05	0.05	0.06	0.04	0.04	0.05	0.03
Apache Parquet	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Apache Tajo	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02
DSpace	0.005	0.005	0.005	0.006	0.005	0.006	0.005	0.006
Google Auto	0.003	0.006	0.003	0.004	0.003	0.006	0.003	0.007
Google Closure	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
Google Guava	0.10	0.11	0.11	0.11	0.10	0.11	0.11	0.11
GSDTSR	1.48	1.82	1.52	1.63	1.56	1.58	1.53	1.57
IOF/ROL	0.01	0.04	0.01	0.01	0.01	0.03	0.01	0.01
Mybatis	0.08	0.10	0.16	0.17	0.18	0.18	0.16	0.18
Paint Control	0.01	0.04	0.02	0.03	0.02	0.14	0.02	0.03
Rails	0.03	0.06	0.04	0.04	0.04	0.06	0.04	0.04

becoming more and more sparse. The high frequent iteration of the test cases increases the length of historical information, but the low-failure test aggravates the sparsity, which leads to sparse rewards of RL and ultimately leads to a generous return of TCP effects, such as Paint Control reduced from 66.73% to 23.41% in NAPFD. Therefore, although *HFD* can solve the sparse reward problem, it cannot measure the impact of recent failure on the test, thereby reducing the TCP effect.

- *AFP* considers the impact of recent failure and uses a larger value to reward from the perspective of historical failure position. *AFP* is used to solve the sparse reward problem. From the convergence perspective, *AFP* improves the standard deviation of NAPFD with more than half of the data sets, especially the standard deviation of eight data sets compared with *APHF*, as shown in the marked of Table 2. Compared to *TF* and *APHF*, there is a significant difference in the TCP effect of *AFP* in Tables C2 and 3. Therefore, the sparse distribution of historical failure positions has a significant impact on the TCP effect. With the historical information-based reward method(*APHF* and *AFP*), in the data sets with high frequent integration such as Google Guava, *AFP* increases the average TCP effect from 23.51% to 27.54% in NAPFD. In the continuous integration process, the failure position increases with the iteration of the test case, and the *AFP* value of the frequent iterative test case is changed by the recent failure position. Therefore, *AFP* can increase the priority of failure test cases in a short period, which is a significant difference from other rewards.
- The existing rewards measure the fault detection ability based on the historical failure of the test case. For the test cases with no failures, it is impossible to measure the impact on follow-up test failures. From the perspective of test requirements, *TOF*-based additional reward is an auxiliary measurement of the fault detection capability of the test case. The combination of reward and additional reward is called the reward with additional reward, where the reward can be any reward method. It can not only measure the fault detection ability of the test case with historical failure but also measure the failure effect of the test case without historical failure. The reward with additional reward mainly increases the reward objects through the judgment of additional reward value *TOF*, to solve the sparse reward problem. By comparing with the reward without additional reward, the TCP effect can be effectively improved on most data sets in Table 4, and the significant difference is also obvious under $\alpha = 0.01$ in Tables C4 and 5. The increase of reward is analyzed by the *TOF* threshold. Under an appropriate threshold, the TCP effect can be improved by increasing reward values and reward objects in RL-based CI testing.

4.7 | Threats to validity

4.7.1 | Internal validity

The main threat to validity is random effects in experiments. First, we used the actual industrial data sets to conduct RL-based CI testing experiments. In the industrial data sets, the modification of the CI module is unknown, which correspondingly leads to the uncertainty of the test case submission and the test results. These factors greatly affect the TCP effect. Second, the combination of neural networks and RL will lead to the instability of the algorithm.[12] In the traditional TCP field, 30 repeated experiments are usually performed to eliminate the influence of random factors on the experimental results. However, according to the experimental results of Spieker et al,[11] the use of RL methods in CI testing usually requires learning 60 cycles, which makes the learned TCP strategy better than heuristic methods. According to the balanced learning model of RL in exploration and utilization, random factors account for a large proportion in the TCP strategy until the 60th integration cycle. Therefore, this article increases the number of repeated experiments to further eliminate the random exploration influence of RL in the early CI testing stage. Each experiment is repeated 60 times to reduce the impact of random exploration.

The other threat to validity is the redundancy of historical information. CI testing has the characteristics of highly frequent iteration, which makes the historical information of test cases increase dramatically. The early failure information of the test case gradually decreases the impact on the last test failures. Considering the relevance of failure test to test failure, we select the test case's recent historical information for feature extraction, which is called a time window-based reward method.[14] The time window size determines the range of recent historical information for feature extraction, which affects the calculation of rewards. Based on the previous research results,[13,14] we selected the time window size with five for related experiments, where the size was obtained through tunning experiments.

4.7.2 | Construct validity

The threat to constructional validity involves the relationship between theory and observation. We used four metrics (NAPFD, Recall, TTF, and Time Consumption) to evaluate the performance of the rewards in RL-based CI testing. Then, the performance was compared among different rewards, including *TF*, *APHF*, *HFD*, *AFP*, and these rewards with *TOF*-based additional reward. These metrics were selected based on prior works, which can effectively evaluate the TCP effect of CI testing. However, these evaluation indicators are not the only metrics to evaluate the TCP quality of CI testing, and different metrics may produce different results.

For the multiple failure executions in the one integration cycle, the execution results are identified according to the failure execution. These failures are considered to be failed together. However, we do not necessarily determine whether these tests are failed with the same faults. We do not consider the correlation among different test cases that detect the same fault. For a test case that detects multiple faults in one cycle, we think that the test case detects multiple faults of different modules in the integration cycle. This situation is only for the merge integration mode, and we do not go into details here. That is to say, we believe that even tests that fail together with different faults are interesting, this kind of common failure may indicate implicit or indirect interaction among the tests.

4.7.3 | External validity

Threats to external validity involve the generalization of our findings. We studied the shared projects from different sources, involving data sets of different sizes. However, the study of data sets with different frequency and failure rate characteristics is not sufficient, which may lead to the unsuitability of the reward method. As discussed before, for the reward without additional reward, although *APHF* is the best reward based on historical failure information overall, the optimal reward is different for different data sets. Accordingly, the sparse rewards of RL can be improved with additional reward for most data sets under RL-based CI testing. However, there is still incompatibility in some data sets, especially the data sets with high failure rates. Therefore, more data sets with the same characteristics are needed to verify the effectiveness of the reward method.

4.7.4 | Conclusion validity

The threat to the conclusion validity is the repeated measures. Repeated experiments are used to eliminate the randomness, which is caused by random exploration in the initial stage of RL. In the process of evaluating the results, the average results of repeated experiments can reflect the experimental results to a certain extent, which might bias the results from the statistical analysis. Therefore, the Turkey HSD is introduced to overcome this limitation, which proved the significant improvement of TCP effect.

5 | CONCLUSIONS AND FUTURE WORK

In the actual industrial program, the high-frequency iteration and low-failure test in CI testing lead to the low reward value and the sparse reward objects in RL-based CI testing. This is the sparse reward problem in RL, which causes RL to be slow or even difficult to converge. This paper focuses on the sparse rewards of RL-based CI testing and studies the solutions from the perspective of increasing the reward value and the reward objects.

In terms of reward value, there are two historical failure information-based rewards designed, including historical failure density(*HFD*) and average failure position(*AFP*). *HFD* objectively describes the historical failure sparsity of the test case. For some sparse data sets, *HFD* improves the reward sparseness and the TCP effect effectively. *AFP* increases the priority of the failure test cases according to the recent failure positions, which greatly affects the priority of accidental failure test cases in the test sequence. *HFD* and *AFP* reduce the impact of sparse rewards by increasing the reward value, where the TCP effect can get improved in RL-based CI testing.

In terms of reward objects, we propose an additional reward method based on test occurrence frequency(*TOF*). *TOF* can be combined with any reward method to improve the sparse rewards of RL-based CI testing. *TOF* is a metric based on the global perspective of the execution history. The reward with additional reward selects reward objects by analyzing the failure effect of the test case, where the failure of the test case will affect the system. On the basis of rewarding failure test cases, the additional reward increases the reward objects of some passed test cases. In addition, the additional reward supplements the reward value for test cases, which have not failed or accidentally failed. The reward with additional reward can effectively improve the sparse reward of RL through the increase of reward objects and the promotion of reward value. The reward with additional reward can get a better TCP effect with the comparison to the reward without additional reward in RL-based CI testing.

The sparse reward is common in RL-based CI testing. In the future, more information, such as requirement-related information and code-related information, can be used to design the reward. In addition, the sparse reward can be optimized based on the agent, such as introducing an experience playback mechanism and introducing new algorithms to optimize the agent's exploration and utilization mechanism.

ACKNOWLEDGMENT

The work described in this paper is supported by the National Natural Science Foundation of China under Grant Nos. 61872026, 61902015, and 62077003.

CONFLICT OF INTEREST

The authors declare no potential conflict of interest.

AUTHOR CONTRIBUTIONS

Yang Yang implemented part of experiments and wrote the original draft. Zheng Li supervised the research and contributed significantly to writing, review and editing. Ying Shang performed the data analysis and validation and helped with constructive discussions. Qianyu Li performed the part of experiments and collected the data.

SUPPORTING INFORMATION

The work described in this paper is supported by the National Natural Science Foundation of China under Grant Nos. 61872026, 61672085, and 61702029, and the Fundamental Research Funds for the Central Universities (XK1802-4).

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

ORCID

Yang Yang https://orcid.org/0000-0001-9257-4631
Zheng Li https://orcid.org/0000-0002-3938-7033
Ying Shang https://orcid.org/0000-0003-4575-0801
Qianyu Li https://orcid.org/0000-0002-7137-999X

ENDNOTES

* https://new.abb.com/products/rpbptics

† https://code.google.com/p/google-shared-dataset-of-test-suite-results/

‡ https://travis-ci.org/rails/rails

§ https://travistorrent.testroots.org/buildlogs/20-12-2016/

¶ https://bitbucket.org/helges/atcs-data

Implementation available at https://bitbucket.org/HelgeS/retecs/src/master/

REFERENCES

1. Vasilescu B, Yu Y, Wang H, Devanbu P, Filkov V. Quality and productivity outcomes relating to continuous integration in github. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering; 2015:805-816.
2. Hilton M, Nelson N, Tunnell T, Marinov D, Dig D. Trade-offs in continuous integration: assurance, security, and flexibility. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering; 2017:197-207.
3. Meyer M. Continuous integration and its tools. *IEEE Softw*. 2014;31(3):14-16.
4. Schneider R. Continuous integration: improving software quality and reducing risk. *Softw Qual Prof*. 2008;10(4):51.
5. Rodriguez P, Haghighatkhah A, Lwakatare LE, Teppola S, Suomalainen T, Eskeli J, Karvonen T, Kuvaja P, Verner JM, Oivo M. Continuous deployment of software intensive products and services: a systematic mapping study. *J Syst Softw*. 2017;123:263-291.
6. Zhang L, Zhou J, Hao D, Zhang L, Mei H. Prioritizing JUnit test cases in absence of coverage information. In: 2009 IEEE International Conference on Software Maintenance; 2009:19-28.
7. Mukherjee R, Patnaik KS. A survey on different approaches for software test case prioritization. Journal of King Saud University-Computer and Information Sciences, 1–14; 2018.
8. Wong WE, Horgan JR, London S, Bellcore HA. A study of effective regression testing in practice; 1997.
9. Bajaj A, Sangwan OP. A systematic literature review of test case prioritization using genetic algorithms. *IEEE Access*. 2019;7:126355-126375.
10. Elbaum S, Rothermel G, Kanduri S, Malishevsky AG. Selecting a cost-effective test case prioritization technique. *Softw Qual J*. 2004;12(3):185-210.
11. Spieker H, Gotlieb A, Marijan D, Mossige M. Reinforcement learning for automatic test case prioritization and selection in continuous integration. In: Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis; 2017:12-22.
12. Sutton RS, Barto AG. *Reinforcement Learning: An Introduction*: MIT press; 2018.
13. Yang Y, Li Z, He L, Zhao R. A systematic study of reward for reinforcement learning based continuous integration testing. *J Syst Softw*. 2020;170:110787.
14. Wu Z, Yang Y, Li Z, Zhao R. A time window based reinforcement learning reward for test case prioritization in continuous integration. In: Proceedings of the 11th Asia-Pacific Symposium on Internetware; 2019:1-6.
15. Kaifeng Z, Yang Y. Methodologies for imitation learning via inverse reinforcement learning: a review. *J Comput Res Dev*. 2019;56(2):254.
16. Dewey D. Reinforcement learning and the reward engineering principle. In: 2014 AAAI Spring Symposium Series; 2014:13-16.
17. Tang H, Houthooft R, Foote D, Stooke A, Chen X, Duan Y, Schulman J, De Turck F, Abbeel P. Exploration: a study of count-based exploration for deep reinforcement learning. In: 31st Conference on Neural Information Processing Systems (NIPS), Vol. 30; 2017:1-18.

18. Sukhbaatar S, Lin Z, Kostrikov I, Synnaeve G, Szlam A, Fergus R. Intrinsic motivation and automatic curricula via asymmetric self-play. arXiv: Learning, https://arxiv.org/abs/1703.05407; 2017.
19. Campos J, Arcuri A, Fraser G, Abreu R. Continuous test generation: enhancing continuous integration with automated test generation. In: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering; 2014:55-66.
20. Ying S, Qianyu L, Yang Y, Zheng L. Occurrence frequency and all historical failure information based method for TCP in CI. In: Proceedings of the International Conference on Software and System Processes; 2019:105-114.
21. Chi J, Qu Y, Zheng Q, Yang Z, Liu T. Relation-based test case prioritization for regression testing. *J Syst Softw*. 2020;163:110539.
22. Jahan H, Feng Z, Mahmud SMH, Dong P. Version specific test case prioritization approach based on artificial neural network. *J Intell Fuzzy Syst*. 2019; 36(6):6181-6194.
23. Mahdieh M, Mirian-Hosseinabadi S-H, Etemadi K, Nosrati A, Jalali S. Incorporating fault-proneness estimations into coverage-based test case prioritization methods. *Inf Softw Technol*. 2020;121:106269.
24. Srikanth H, Hettiarachchi C, Do H. Requirements based test prioritization using risk factors. *Inf Softw Technol*. 2016;69:71-83.
25. Cho Y, Kim J, Lee E. History-based test case prioritization for failure information. In: 2016 23rd Asia-Pacific Software Engineering Conference (APSEC); 2016:385-388.
26. Li Z, Harman M, Hierons RM. Search algorithms for regression test case prioritization. *IEEE Trans Softw Eng*. 2007;33(4):225-237.
27. Li Z, Bian Y, Zhao R, Cheng J. A fine-grained parallel multi-objective test case prioritization on GPU. In: International Symposium on Search Based Software Engineering; 2013:111-125.
28. Bian Y, Li Z, Zhao R, Gong D. Epistasis based ACO for regression test case prioritization. *IEEE Trans Emerg Top Comput Intell*. 2017;1(3):213-223.
29. Yu L, Xu L, Tsai WT. Time-constrained test selection for regression testing. In: International Conference on Advanced Data Mining and Applications; 2010:221-232.
30. Souza LSD, Miranda PBCD, Prudencio RBC, Barros FDA. A multi-objective particle swarm optimization for test case selection based on functional requirements coverage and execution effort. In: IEEE International Conference on Tools with Artificial Intelligence; 2011:245-252.
31. Alkawaz MH, Silvarajoo A. A survey on test case prioritization and optimization techniques in software regression testing; In: Proceedings of the 7th Conference on Systems, Process and Control (ICSPC). 2019.
32. Ledru Y, Petrenko A, Boroday S, Mandran N. Prioritizing test cases with string distances. *Autom Softw Eng*. 2012;19(1):65-95.
33. Kim J-M, Porter A. A history-based test prioritization technique for regression testing in resource constrained environments. In: Proceedings of the 24th international conference on software engineering; 2002:119-129.
34. Marijan D, Gotlieb A, Sen S. Test case prioritization for continuous regression testing: an industrial case study. In: IEEE International Conference on Software Maintenance; 2013:540-543.
35. Elbaum S, Rothermel G, Penix J. Techniques for improving regression testing in continuous integration development environments. In: Acm Sigsoft International Symposium; 2014:235-245.
36. Srikanth H, Williams L. On the economics of requirements-based test case prioritization. *ACM Sigsoft Softw Eng Notes*. 2005;30(4):1-3.
37. Strandberg PE, Sundmark D, Afzal W, Ostrand TJ, Weyuker EJ. Experience report: automated system level regression test prioritization using multiple factors. In: 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE); 2016:12-23.
38. Do H, Rothermel G. An empirical study of regression testing techniques incorporating context and lifetime factors and improved cost-benefit models. In: Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering; 2006:141-151.
39. Lu Y, Lou Y, Cheng S, Zhang L, Hao D, Zhou Y, Zhang L. How does regression test prioritization perform in real-world software evolution? In: Proceedings of the 38th International Conference on Software Engineering; 2016:535-546.
40. Levine S, Pastor P, Krizhevsky A, Ibarz J, Quillen D. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The Int J Robotics Res*. 2016;37(4-5):421-436.
41. Isele D, Rahimi R, Cosgun A, Subramanian K, Fujimura K. Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. In: 2018 IEEE International Conference on Robotics and Automation (ICRA); 2018:2034-2039.
42. Andrychowicz M, Wolski F, Ray A, Schneider J, Fong R, Welinder P, McGrew B, Tobin J, Abbeel P, Zaremba W. Hindsight experience replay. arXiv preprint arXiv:170701495, https://arxiv.org/abs/1707.01495; 2017.
43. Lanka S, Wu T. Archer: Aggressive rewards to counter bias in hindsight experience replay. arXiv preprint arXiv:180902070, https://arxiv.org/abs/1809.02070; 2018.
44. Lima JAP, Vergilio SR. Test case prioritization in continuous integration environments: A systematic mapping study. *Inf Softw Technol*. 2020;121: 106268.
45. Qu X, Cohen MB, Woolf KM. Combinatorial interaction regression testing: a study of test case generation and prioritization. In: IEEE International Conference on Software Maintenance; 2007:255-264.
46. Rothermel G, Untch RH, Chu C, Harrold MJ. Prioritizing test cases for regression testing. *IEEE Trans Softw Eng*. 2001;27(10):929-948.

How to cite this article: Yang Y, Li Z, Shang Y, Li Q. Sparse reward for reinforcement learning-based continuous integration testing. *J Softw Evol Proc*. 2021;e2409. doi:10.1002/smr.2409

APPENDIX A: The Analysis of Variance

The Analysis of Variance (ANOVA) is introduced with *F* and *P* values to compare the significant difference between the two data sets. *F* value is the ratio of two mean squares, where the significance is more obvious with a greater *F* value. *P* value is a parameter for judging the result of the hypothesis test, where the significant result is remarkable with a smaller *P* value. For the data with no standard deviation distance from the average, a valid *F* value cannot be obtained, which will be marked with NULL. The higher the *F* value, the more likely it is that the hypothesis of two data sets coming from the same source will be rejected. *P* value is further used to measure significant differences in ANOVA, where $\alpha = 0.01$.

A.1 | The ANOVA of the historical information-based reward

In order to verify the significant difference of historical information-based reward, *F* and *P* values are compared in Tables C1 and C2. *P* value is calculated based on *F* value. Table C1 lists the *F* values of all data sets with different reward functions, where F_1 is compared to TF and F_2 is compared to APHF. Table C2 lists the *P* values of all data sets with P_1 compared to TF and P_2 compared to APHF, where the gray background indicates the values with significant differences.

There are two test hypotheses:

H0: There is no difference between the TCP effects of the two reward functions.

H1: There is a significant difference between the TCP effects of the two reward functions.

Under the inspection level of 0.01, the significant result is marked with gray background in Table C2. Except for NULL, there are 53.13% significant difference values in total.

With the NAPFD comparison of *HFD* and *TF*, there is a significant difference in nine data sets. With the NAPFD comparison of *AFP* and *TF*, there is a significant difference in nine data sets. With the NAPFD comparison of *HFD* and *APFH*, there is a significant difference in eight data sets. With the NAPFD comparison of *AFP* and *APHF*, there is a significant difference in seven data sets. In most data sets, through the solution of sparse rewards, the TCP metrics are significantly different.

Based on the analysis of the *P* value, more than half of the significant results indicate that there is a higher probability of rejecting the *H*0 hypothesis. Therefore, the *H*1 hypothesis holds, which means the TCP effect is improved significantly by the rewards of *HFD* and *AFP*.

A.2 | The ANOVA of the reward with additional reward

The ANOVA of the reward with additional reward and the reward without additional reward is compared with *F* and *P* values. Table C3 lists the *F* value of the reward with additional reward compared to the reward without additional reward. The higher the *F* value, the more likely it is that the hypothesis of two data sets coming from the same source will be rejected. For the data with no standard deviation distance from the average, a valid *F* value cannot be obtained, and it is marked with NULL. In Table C4, *P* value is further used to measure significant differences in ANOVA, where $\alpha = 0.01$. Table C4 lists the *P* values of all data sets, which compares the reward with additional reward to the reward without additional reward.

There are two test hypotheses:

H0: There is no difference between the TCP effects of the reward with additional reward and the reward without additional reward.

H1: There is a significant difference between the TCP effects of the reward with additional reward and the reward without additional reward.

Under the inspection level of 0.01, the significant result is marked with gray background in Table C4. In the gray background, the TCP effect of the reward with additional reward is significantly different from the TCP effect of the reward without additional reward. Except for NULL, there are 44.58% significant difference values in total.

We take NAPFD as an example. With the comparison of *TF_Add* and *TF*, there are significant differences in seven data sets. With the comparison of *APHF_Add* and *APHF*, there are significant differences in six data sets. With the comparison of *HFD_Add* and *HFD*, there are significant differences in six data sets. With the comparison of *AFP* and *AFP_Add*, there are significant differences in eight data sets. Therefore, in most data sets, there are significant differences in TCP metrics between the reward with and without additional reward, which rejects the H0 hypothesis.

TABLE A1 *F* value analysis of different rewards comapred to TF and APHF

	NAPFD				Recall				TTF			
	HFD		AFP		HFD		AFP		HFD		AFP	
Data set	F_1	F_2	F_1	F_2	F_1	F_2	F_1	F_2	F_1	F_2	F_1	F_2
Apache Commons	1.83E+00	6.28E−01	1.07E+00	3.67E−01	NULL	NULL	NULL	NULL	1.76E+00	1.32E+00	1.01E+00	7.63E−01
Apache Drill	2.75E+00	2.04E+00	5.19E−01	3.85E−01	5.28E−01	2.23E−01	8.40E−01	3.55E−01	2.65E+00	1.03E+01	7.35E−01	2.86E+00
Apache Hive	1.67E+00	1.27E+00	4.40E−01	3.36E−01	1.28E+00	1.02E+00	8.99E−02	7.20E−02	9.28E−01	9.85E−01	1.57E−01	1.66E−01
Apache Parquet	3.00E+00	3.90E−01	6.05E+00	7.85E−01	7.89E+26	1.54E−02	2.62E+27	5.09E−02	2.52E+00	4.03E−02	3.75E+00	6.00E−02
Apache Tajo	2.16E+00	3.57E+00	2.04E+00	3.37E+00	1.79E+00	1.00E+00	1.79E+00	1.00E+00	4.07E−02	2.88E−02	9.23E−01	6.52E−01
DSpace	1.91E+00	2.59E−01	4.57E+00	6.20E−01	1.58E+00	2.40E+00	9.80E−01	1.48E+00	1.36E+00	4.54E−02	8.85E+00	2.95E−01
Google Auto	1.89E+28	4.88E−01	9.00E+00	2.33E−28	NULL	0.00E+00	NULL	0.00E+00	1.09E+28	7.30E−01	2.50E−01	2.50E−01
Google Closure	1.95E+01	6.32E+03	1.07E+01	3.46E+03	9.00E+00	2.97E+27	1.09E+01	3.58E+27	1.64E+02	1.62E+02	1.96E+00	1.96E+00
Google Guava	2.32E−02	2.69E−02	1.61E+00	1.86E+00	7.02E+00	8.13E+00	1.93E+00	2.24E+00	1.55E+00	1.93E+00	1.55E+00	1.93E+00
GSDTSR	4.07E−01	1.26E−02	4.35E+01	1.34E+00	5.66E−01	2.00E−02	3.84E+01	1.36E+00	1.02E−01	5.85E−02	2.18E+00	1.25E+00
IOF/ROL	2.62E−01	1.12E+01	2.31E+00	9.89E+01	2.75E−01	8.67E+00	1.96E+00	6.17E+01	6.72E−01	2.05E+02	1.94E+00	5.91E+02
Mybatis	4.16E+00	8.63E−01	4.04E+00	8.39E−01	1.38E+00	9.83E−01	1.00E+00	7.14E−01	1.73E+01	8.65E−01	2.06E+01	1.03E+00
Paint Control	7.62E+00	1.26E+01	2.51E+00	4.16E+00	9.71E+00	1.88E+01	1.23E+00	2.38E+00	2.05E+00	7.10E−01	1.27E+01	4.42E+00
Rails	3.26E+00	2.62E+02	7.90E−03	6.35E−01	2.92E+00	2.10E+02	5.37E−03	3.87E−01	6.55E+00	3.88E+02	1.77E−02	1.05E+00

Note: F_1 is the ANOVA compared to *TF*, and F_2 is the ANOVA compared to *APHF*.

TABLE A2 *P* value of different rewards compared to TF and APHF ($\alpha = 0.01$)

	NAPFD				Recall				TTF			
	HFD		AFP		HFD		AFP		HFD		AFP	
Data set	P_1	P_2	P_1	P_2	P_1	P_2	P_1	P_2	P_1	P_2	P_1	P_2
Apache Commons	4.74E−01	2.43E−01	9.69E−01	6.53E−01	NULL	NULL	NULL	NULL	4.57E−01	6.77E−02	9.11E−01	3.15E−01
Apache Drill	6.72E−03	4.46E−02	1.19E−01	9.79E−01	9.66E−01	2.49E−07	1.26E−01	2.06E−04	5.11E−01	5.76E−08	2.74E−01	2.86E−17
Apache Hive	8.16E−01	9.10E−01	1.83E−03	3.24E−02	8.60E−01	3.98E−01	5.35E−06	1.90E−03	5.64E−01	1.51E−01	3.73E−04	3.26E−03
Apache Parquet	5.64E−22	2.52E−06	2.28E−08	5.47E−01	3.81E−03	9.61E−02	8.37E−01	2.21E−01	2.39E−22	8.39E−01	1.97E−04	2.11E−02
Apache Tajo	1.89E−34	1.68E−03	3.00E−35	1.07E−03	2.81E−07	2.00E−06	6.62E−01	2.61E−01	1.81E−01	1.82E−05	5.71E−05	7.43E−01
DSpace	8.99E−17	8.47E−17	8.57E−29	1.48E−02	2.80E−01	8.63E−05	2.47E−04	1.42E−02	5.72E−29	6.19E−45	1.26E−58	8.36E−01
Google Auto	9.56E−17	8.62E−01	NULL	7.84E−13	NULL	3.21E−01	NULL	3.21E−01	1.92E−16	4.77E−01	NULL	4.08E−15
Google Closure	2.84E−01	1.61E−01	7.35E−02	2.19E−02	3.76E−01	1.59E−01	7.15E−02	2.05E−02	1.22E−01	1.54E−01	3.87E−01	5.17E−02
Google Guava	3.38E−06	7.19E−07	5.75E−03	1.73E−03	1.35E−02	1.01E−02	5.83E−04	1.47E−01	2.51E−02	2.61E−02	1.25E−02	3.30E−03
GSDTSR	6.64E−01	3.78E−24	1.25E−16	5.48E−03	9.39E−01	3.90E−24	1.18E−16	5.47E−03	1.89E−02	6.62E−27	9.58E−19	7.98E−02
IOF/ROL	3.80E−05	1.34E−58	4.78E−10	3.00E−19	2.36E−05	6.35E−55	7.09E−09	7.63E−19	5.23E−04	9.23E−39	5.60E−06	7.89E−13
Mybatis	3.40E−04	5.45E−02	1.76E−07	9.48E−01	9.15E−01	9.01E−01	9.84E−01	8.25E−01	1.46E−07	6.50E−03	5.25E−13	8.70E−01
Paint Control	3.01E−44	3.71E−46	5.47E−01	4.26E−03	3.28E−41	2.07E−43	1.75E−01	1.33E−02	2.93E−52	8.44E−45	1.23E−03	8.29E−03
Rails	8.10E−01	4.05E−15	1.11E−26	1.84E−05	5.44E−01	1.79E−14	5.03E−27	4.20E−07	1.65E−02	7.69E−13	5.70E−24	1.14E−03

Note: P_1 is the ANOVA compared to *TF*, and P_2 is the ANOVA compared to *APHF*

TABLE A3 *F* value of different rewards with additional reward compared to the rewards without additional reward

	NAPFD				Recall				TTF			
Data set	TF_Add	APHF_Add	HFD_Add	AFP_Add	TF_Add	APHF_Add	HFD_Add	AFP_Add	TF_Add	APHF_Add	HFD_Add	AFP_Add
Apache Commons	3.31E+01	1.14E+01	3.36E+00	5.27E+01	NULL	NULL	NULL	NULL	1.13E+03	3.08E+02	7.74E+00	9.34E+02
Apache Drill	1.02E+01	2.34E+00	2.34E+00	2.92E+00	5.32E+00	2.03E+00	1.28E+01	2.43E+00	8.84E−01	2.60E+00	2.25E−01	1.23E+00
Apache Hive	7.03E+00	7.40E+00	3.67E+00	1.12E+01	1.43E+00	8.08E−01	6.01E−01	3.84E+00	4.49E−01	1.16E+00	9.75E−01	6.40E+00
Apache Parquet	8.18E−01	5.50E−01	1.35E+00	6.81E−01	9.10E+26	8.27E−03	1.15E+00	1.55E+00	1.62E−01	9.74E−02	5.06E−01	1.74E+00
Apache Tajo	2.58E−01	1.11E+00	1.25E+00	5.16E−01	9.58E−04	1.00E+00	1.00E+00	1.00E+00	7.61E−03	3.44E−01	1.37E+00	1.33E+00
DSpace	2.71E−02	2.20E+00	7.95E−01	1.26E+00	8.01E−28	7.19E−01	1.85E−01	1.95E−01	8.06E−03	9.92E−01	5.92E−01	5.84E−01
Google Auto	2.73E+28	6.74E−01	4.90E−01	2.19E+25	NULL	0.00E+00	NULL	NULL	1.10E+26	1.01E+00	4.80E−01	5.62E+26
Google Closure	2.48E+01	1.22E+04	1.01E+00	1.77E+00	2.49E+01	1.26E+28	1.39E+00	1.74E+00	1.85E+00	3.71E+00	5.58E−01	6.19E−01
Google Guava	1.67E+00	1.88E+00	1.49E+00	9.62E−01	2.90E+01	2.37E+00	4.27E−01	9.90E−01	5.02E+00	1.89E+00	6.74E−01	9.18E−01
GSDTSR	1.78E+01	6.25E−01	6.60E+01	5.36E−01	1.74E+01	6.48E−01	4.49E+01	5.54E−01	2.35E+00	8.20E−01	2.49E+01	6.14E−01
IOF/ROL	1.93E+00	1.32E+00	2.59E+00	7.91E−01	1.78E+00	1.39E+00	2.51E+00	8.00E−01	1.37E+00	2.21E+00	1.30E+00	5.23E−01
Mybatis	4.37E+00	2.57E−01	1.46E+00	9.46E−02	2.90E+00	2.72E+00	2.71E+01	7.78E−02	7.25E+00	3.66E+00	2.28E+01	1.46E−01
Paint Control	1.62E−01	5.52E−01	1.76E+00	1.64E−01	2.21E−01	4.97E−01	1.32E+00	1.42E−01	4.70E−01	2.58E−01	1.54E+01	3.57E−01
Rails	9.90E−01	4.28E−01	1.69E−01	7.87E−01	9.89E−01	4.43E−01	2.10E−01	8.10E−01	1.04E+00	7.33E−01	5.30E−02	3.32E+00

TABLE A4 *P* value of different rewards with additional reward compared to the rewards without additional reward ($\alpha = 0.01$)

	NAPFD				Recall				TTF			
Data set	TF_Add	APHF_Add	HFD_Add	AFP_Add	TF_Add	APHF_Add	HFD_Add	AFP_Add	TF_Add	APHF_Add	HFD_Add	AFP_Add
Apache Commons	2.62E−01	7.92E−01	1.74E−01	2.26E−01	1.09E−03	4.12E−02	4.45E−02	4.12E−03	5.58E−04	4.74E−02	9.68E−02	1.62E−03
Apache Drill	1.43E−25	4.47E−19	4.60E−27	3.87E−22	1.83E−20	5.08E−12	1.73E−20	3.33E−14	5.52E−04	6.03E−05	1.73E−03	9.37E−01
Apache Hive	2.33E−06	1.44E−07	1.68E−08	7.43E−09	3.64E−01	1.09E−02	2.17E−01	2.17E−01	2.71E−09	1.42E−03	1.72E−06	4.18E−14
Apache Parquet	2.99E−04	9.01E−01	6.58E−01	3.47E−01	1.00E+00	1.39E−01	4.45E−02	1.40E−02	1.27E−05	9.10E−02	1.61E−02	1.93E−04
Apache Tajo	9.88E−10	5.96E−03	3.25E−01	9.50E−01	5.16E−05	3.03E−02	5.91E−01	6.56E−01	3.13E−04	1.20E−01	2.51E−01	7.76E−01
DSpace	5.20E−01	3.44E−02	4.92E−01	3.98E−04	3.54E−03	8.31E−01	9.42E−01	2.45E−05	2.53E−01	1.47E−01	5.24E−01	1.20E−04
Google Auto	1.46E−01	8.12E−01	2.69E−02	2.04E−05	8.32E−02	3.21E−01	NULL	NULL	1.29E−04	8.79E−01	2.48E−02	2.04E−05
Google Closure	4.01E−02	1.39E−02	9.99E−01	3.85E−01	4.21E−02	1.38E−02	8.80E−01	3.90E−01	9.28E−01	8.88E−02	7.06E−01	7.31E−01
Google Guava	3.50E−02	6.17E−04	1.13E−01	3.36E−01	4.52E−01	5.35E−04	8.12E−01	1.97E−01	2.09E−03	5.49E−04	9.54E−01	4.57E−01
GSDTSR	4.27E−07	5.49E−12	1.01E−02	9.04E−05	4.64E−07	7.66E−12	1.61E−02	9.41E−05	2.01E−08	2.02E−11	6.63E−05	1.08E−05
IOF/ROL	6.24E−06	1.09E−01	2.34E−03	3.69E−05	1.12E−05	7.51E−02	2.01E−03	1.70E−04	6.86E−04	9.62E−03	4.06E−04	2.09E−01
Mybatis	1.84E−14	1.69E−05	6.71E−03	5.85E−08	7.13E−01	7.96E−01	1.11E−02	4.51E−01	1.42E−35	2.71E−04	3.81E−06	2.19E−09
Paint Control	1.23E−01	1.23E−01	8.09E−14	4.43E−03	1.28E−01	2.73E−01	6.10E−17	2.57E−03	1.41E−01	2.03E−02	1.62E−01	2.58E−02
Rails	3.98E−01	5.28E−01	1.58E−07	1.41E−01	5.29E−01	7.59E−01	1.61E−06	3.42E−01	1.34E−01	8.36E−01	2.28E−08	2.10E−02