Dynamic Random Testing: Technique and Experimental Evaluation

Hanyu Pei, Kai-Yuan Cai, Beibei Yin, Aditya P. Mathur, and Min Xie

***Abstract*—A particularly good software testing strategy is to achieve the underlying testing goal while solving the problems of tradeoffs between testing effectiveness and efficiency. To improve the fault detection effectiveness of software testing, the principle of feedback control theory was adopted, which motivated the proposal of dynamic random testing (DRT). The main idea behind DRT is using the testing results to guide the test case selection to increase the selection probabilities of the subdomains with higher fault detection rates. Previous works show that DRT strategy can achieve better effectiveness than random testing strategy and random partition testing strategy, and has significantly lower computational costs than adaptive testing strategy. However, the essential factors that affect the performance of DRT, i.e., adjusting parameters, initial profile, and test case classification have not been thoroughly investigated. Besides, some experimental assumptions are inconsistent with real scenarios. Therefore, this paper gives a series of investigations on DRT with a set of practical subject programs. More specifically, the effectiveness and efficiency of DRT are presented, and the extended experiments on DRT with relevant factors are conducted. The results indicate that the effectiveness of DRT is robust to different initial profiles and affected noticeably by the adjusting parameter settings and test case classification methods.**

***Index Terms*—Dynamic random testing (DRT), random partition testing (RPT), software cybernetics, testing profiles.**

NOMENCLATURE

Acronyms and Abbreviations

RT	Random testing.
PT	Partition testing.
RPT	Random partition testing.
DRT	Dynamic random testing.
O-DRT	Optimization dynamic random testing.
AT	Adaptive testing.
ART	Adaptive random testing.
APFD	Average percentage of faults detected.
Diff*D*	Difference degree.

Notation

C_i	*i*th subdomain of test suite.
P_i	Selection probability of subdomain *i*.
ε, δ	Adjusting parameters.
θ	Fault detection rate of subdomain.
T	Total number of test cases consumed to detect all seeded faults.
D	Standard variance of T over a given number of trials.
CV	Coefficients of variation over a given number of trials.
TF_i	Smallest number of test cases in the sequence that needs to be executed to detect fault *i*.
V_i	Variance of APFD_i.
T-measure	Number of test cases executed to detect a fixed number of faults.
F-measure	Number of test cases executed to detect the first failure.

Manuscript received January 2, 2018; revised June 19, 2018, December 5, 2018, and February 21, 2019; accepted April 9, 2019. This work was supported in part by the National Natural Science Foundation of China under Grant 61402027, in part by the National Natural Science Foundation of China under Grant 61772055, and in part by Equipment preresearch projects under Grant 41402020102. Associate Editor: T. H. Tse. *(Corresponding author: Beibei Yin.)*

H. Pei is with the Department of Automatic Control, School of Automation Science and Electrical Engineering, Beihang University, Beijing 100083, China, and also with the Department of Systems Engineering and Engineering Management, City University of Hong Kong, Hong Kong (e-mail: peihanyu@buaa.edu.cn).

K.-Y. Cai and B. Yin are with the Department of Automatic Control, School of Automation Science and Electrical Engineering, Beihang University, Beijing 100083, China (e-mail: kycai@buaa.edu.cn; yinbeibei@buaa.edu.cn).

A. P. Mathur is with Department of Computer Science, Purdue University, West Lafayette, IN 47907 USA (e-mail: aditya_mathur@sutd.edu.sg).

M. Xie is with the Department of Systems Engineering and Engineering Management, City University of Hong Kong, Hong Kong (e-mail: minxie@cityu.edu.hk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TR.2019.2911593

I. INTRODUCTION

ONE aim of software testing is to find out software faults and improve software reliability. A good software testing strategy seeks to achieve test goals while minimizing resource consumption, which is usually measured by the fault detection effectiveness and computational costs [1]. Due to this issue, a testing technique, dynamic random testing (DRT) is proposed by Cai *et al.* [2]. The DRT technique and its evaluation are the focuses of this paper. The following fundamental questions of DRT will be addressed through a series of experiments.

1) Can DRT achieve better fault detection effectiveness over other testing strategies in actual testing for the software programs of various scales and functionalities?
2) How much computational cost will DRT incur? Does it noticeably affect the efficiency of DRT?
3) What is the impact of the following factors on the performance of DRT, i.e., adjusting parameters, initial profile, and test case classification?

Random testing (RT) and partition testing (PT) are two well-known software testing techniques [3]. In RT, the test suite of

software under test (SUT), which comprises some distinct test cases, is selected according to any given distribution [4]. No partitioning is applied to the test suite. In contrast, in subdomain testing, the test suite of SUT is partitioned into a number of disjoint or overlapping subdomains, and one or more test cases are generated and selected from each subdomain. More specifically, if the subdomains are disjoint, the strategy is also referred to as PT. Various coverage criteria such as functional coverage, statement coverage, and dataflow coverage can be adopted in subdomain testing or PT to classify the test suite [5], [6]. Random partition testing (RPT) [7] is a combination of RT and PT. Suppose that the test suite of SUT is partitioned into m subdomains $\{C_1, C_2, ..., C_m\}$, and the ith subdomain C_i consist of k_i test cases. A test case is selected in two steps. First, a subdomain is selected according to a given testing profile $\{p_1, p_2, ..., p_m\}$, where p_i is the selection probability of ith subdomain. Second, a test case from the chosen subdomain is selected in accordance with a uniform probability distribution, in which the probability of each test case being selected in the ith subdomain is $1/k_i$.

In software testing, the testing results indicate whether faults are detected after the execution of test cases. However, the results containing useful information are usually utilized inadequately in RT and RPT, e.g., the testing profile is kept constant during the testing process such that the testing results have no impact on them. This approach might not be desirable because failure-causing test cases tend to cluster in the input domain: Recent research highlighted the existence of the so-called Pareto rule in the failure distribution, which means that 20 percent of source files tend to include approximately 80 percent of faults [8]. Some studies have also shown that PT will usually be better and never be worse than RT when the test cases in subdomains with higher fault detection rates are selected with higher selection probabilities. The result provides a clear motivation of trying to select more test cases from subdomains with higher failure rates [9]–[11]. Keeping this in mind, Cai *et al.* [12] proposed an adaptive testing (AT) strategy by using the results of testing data as feedback to select test cases in an optimal manner. More specifically, the SUT is modeled as a controlled Markov chain. Then, by estimating the parameters (fault detection rates of subdomains) based on the historical testing data, the software testing strategy can be adjusted online to make an optimal testing decision. This method can minimize the software testing cost by deploying the adaptive control theory. Compared with RPT, the effectiveness of AT is much higher; however, the computational costs required for parameter estimation algorithms, such as least square estimation and the genetic algorithm (GA), are relatively high.

Based on the research works mentioned above, it can be verified that the utilization of the PT and feedback mechanism in the testing strategy can improve fault detection effectiveness. In addition, a challenge associated with new testing strategies is how to improve fault detection effectiveness with a minimal increase in computational cost. This constitutes the motivation behind the proposal of DRT to achieve a balanced state. If test cases with similar characteristics are partitioned into same subdomains, then when a test case in a subdomain reveals a failure, the rest of the test cases in that subdomain have higher probabilities to reveal failures since the failure-causing test cases tend to cluster in the input domain. It means that this subdomain might have higher fault detection rate at that time. Therefore, the main principle of DRT is dynamically adjusting the testing profile to increase the selection probabilities of subdomains with higher fault detection rates based on previous testing results. In DRT, the subdomains $\{C_1, C_2, ..., C_m\}$ and the corresponding probability distributions $\{p_1, p_2, ..., p_m\}$ define a testing profile for the SUT. The testing results are used to adjust the testing profile in a dynamic manner: if a test case of subdomain C_i triggers a failure, then the selection probability of C_i will increase from p_i to $p_i + \varepsilon$ (the upper bound of p_i is 1); otherwise, the selection probability will decrease to $p_i - \delta$ (the lower bound of p_i is 0). The adjusting parameters ε and δ, which can be set in the range of 0 to 1, denote the increment and decrement of the selection probabilities of subdomains. In DRT, the testing profile is adjusted gradually to an optimal one through a feedback mechanism, and the complex optimizing functions related to the selection of test cases can be avoided.

Some analysis of the performance of DRT were included in previous studies [13]–[16], which have shown that DRT can use fewer test cases to detect a fixed number of faults comparing with RT and RPT in most cases. Besides, some improved DRT strategies, such as history-based DRT (DRT-h) [14], adaptive DRT (A-DRT) [15], and optimization DRT (O-DRT) [16] were proposed to further enhance the effectiveness of DRT, which provide potential improving directions on DRT. However, previous studies still need improvement. For example, the prioritization effectiveness (whether a testing strategy can detect faults in the earlier stage) of DRT were not investigated. And time consumption of DRT was only recorded in one subject, which is insufficient to verify the computational cost of DRT. Moreover, some experimental assumptions may also be not consistent with the real scenarios in previous studies on DRT. For one thing, the test suite remains unchanged during the testing process, which might lead to duplicate the selection of the executed test cases. For another thing, the fault detection process from revealing the failure to finding out the faults is not illustrated clearly, which should be explained in the controlled experiment. Besides, several critical factors underlying DRT, such as adjusting parameters, initial profile, and test case classification, have also not been investigated thoroughly. They are preset before the testing process and constitute the principal settings of DRT strategy.

The adjusting parameters, ε and δ, characterize the scale of adjustment of the testing profile after the execution of each test case. Intuitively, the value of adjusting parameters ε and δ should not be too high or too low. If the values are too high, the testing profile will fluctuate drastically; otherwise, it will take a long time for the testing profile to achieve an optimal one. However, in previous studies, only a part of settings of adjusting parameters are investigated. The effect of adjusting parameters on the performance of DRT needs further investigation. Therefore, we need to observe that how will the effectiveness of DRT be affected when ε and δ change, and whether there exists a rule to describe the effect of adjusting parameters?

The initial profile, which is usually built based on previous experience, determines the selection probability of each subdomain at the beginning of testing. In most cases, the initial profile is not the optimal testing profile, especially when lacking information about the software. If the initial profile deviates a lot from the optimal one, will DRT algorithm be effective enough to adjust the profile?

The test suite can be classified according to any criteria. Many classification methods, e.g., functional classification method, combination classification method, and random classification method can be used in DRT to partition the test suite into subdomains, which may lead to different distributions of the fault detection rates of the subdomains. Different classification means that the testing process and the performance of DRT might vary. Hence, it is interesting to explore the sensitivity of DRT to the test case classification.

Therefore, the effectiveness and efficiency of DRT are studied comprehensively and the effect of the above factors are investigated in this paper. First, a DRT strategy with a supplement is proposed to enhance the performance of DRT and make the testing process more pragmatic. The executed test case removal mechanism is combined into the DRT strategy. And the program instrumentation method is used in controlled experiments, which can make the faults traceable and easily being located. Second, a basic experiment is conducted to compare the effectiveness and efficiency of DRT with other testing strategies in terms of various metrics, i.e., *T*-measure, APFD, and *computational cost* (which will be illustrated in Section IV). Then, three extended experiments are conducted to investigate the influence of the three factors, i.e., adjusting parameters, initial profiles, and test case classification, on the performance of DRT.

This paper makes the following main contributions.

1) We investigate the performance of DRT through a more comprehensive study compared with previous works, in which more metrics are adopted in the experiments.
2) The effect of factors, i.e., adjusting parameters, initial profile, and test case classification are analyzed, which gives a guide for their settings and provides possible improvement directions.

The rest of this paper is organized as follows. Section II presents a brief investigation of the related works. Section III is the introduction of DRT with supplements and the factor analysis. Section IV presents the basic experiment conducted on five subjects to compare the effectiveness and efficiency of DRT with that of RT, RPT, and AT. Section V reports three extended experiments, in which the effect of factors is investigated. Section VI illustrates the possible threats to the validity of this paper. Section VII concludes this paper.

II. Related Works

A. Traditional Testing Strategies

RT, PT, and RPT are traditional and classical testing strategies [17] and have received considerable attention in the literature.

Though very simple, RT is still a widely used testing technique, along with other more complicated and systematic testing methods [18]–[20]. A theoretical analysis by Arcuri and Briand [8] has also indicated that RT could be as effective as systematic testing strategies provided that there are no constraints on the features of the software. RT has already been applied successfully in many applications, including database systems, Java Just-In-Time compilers, security assessment, Windows NT, Mac OS robustness assessment, Android applications testing, and unmanned aerial vehicles real-time system [21]–[23]. Another widely adopted testing technique is PT, which assumes the existence of subdomains in the input space. Many well-known testing methods such as statement testing, branch testing, path testing, and certain variants of mutation testing are subdomain-based, i.e., they all require that the test suites are partitioned into nonoverlapping or overlapping subdomains [24], [25].

The comparison of the performance between RT and PT has been conducted in many research works [3], [4], [7], [9], [11], [26], [27]. Duran and Ntafos [28] have conducted simulation experiments to compare RT and PT using different model assumptions and suggested that despite the slightly superior result achieved by using PT and RT appears to be more cost-effective. Hamlet and Taylor [29] have conducted similar simulation experiments and somewhat confirmed their observation. Weyuker and Jeng [26] have compared RT and PT from an analytical point of view and the results have shown that the performance of PT depends on the fault detection ability of the partitions and in the extreme case when the fault detection rates in the subdomains are equal, the performance of PT is always the same as that of RT. And a study conducted by Boland *et al.* [9] has investigated the relative effectiveness of PT over RT. RPT is a combination of RT and PT. On the one hand, RPT partitions the test suite into several subdomains, in which the used classification methods are similar to PT; on the other hand, RPT selects subdomains randomly and then selects a test case from the chosen subdomain, which is intimately related to RT.

A major advantage of RT, PT, and RPT is that the algorithms of these testing strategies are uncomplicated, and they are easily understood and implemented by testing engineers. Hence, they have been used widely in regular testing projects and usually treated as benchmarks for software testing. However, RT ignores the structure information of the software and all these strategies have ignored historical data during the testing process, which has led to several difficulties in improving their performance through internal or external guidance.

B. Improved Testing Strategies

Several testing techniques have been proposed to improve PT, RT, and RPT, i.e., AT and Adaptive Random Testing (ART).

Following the idea of software cybernetics, AT has been proposed by Cai [30]. This method adjusts the selection of test cases online following the idea of adaptive control to achieve an optimization objective, such as minimizing the number of test cases executed to detect and remove multiple faults (referred to as the *T*-measure). For example, in AT based on the genetic algorithm (AT-GA) [12], the test suite is divided into several subdomains that are set based on fitness. First, a subdomain is selected randomly, and a test case in this subdomain is executed. The testing result of whether faults are detected is recorded. Second, the fitness values of all subdomains are calculated

through multi-iterations of intersection and mutation (which are the main techniques in the field of GA. The fitness values will continually change until the number of iterations is achieved. Then, the next test case is selected from the subdomain with the highest fitness value. A case study using the space program has shown that AT-GA outperforms RT and RPT. However, a major concern in the application of AT is its complexity and computational cost during test case selection. Therefore, an AT-RT hybrid approach has been proposed by Lv *et al.* [1], which uses AT and RPT in an alternative manner to select test cases. Experimental results have shown that the hybrid approach considerably reduces the computational cost of the original AT strategy while still outperforming RT and RPT strategies. Besides, the principle of AT strategy has been used in software reliability assessment. AT with gradient descent method (AT-GD) has been proposed by Lv *et al.* [31] to minimize the variance of reliability estimator. The associated theoretical analysis and simulations have indicated that AT-GD converges to the globally optimal solution as the assessment process proceeds.

Compared with traditional testing strategies, AT strategy significantly improves the fault detection effectiveness, but it has very high time consumption. The sophisticated algorithm behind AT hinders the enhancement of efficiency and is hard to understand by engineers. Therefore, the tradeoffs between testing effectiveness and efficiency are a critical issue in this paper, and the AT algorithm is used mainly for algorithm performance comparison.

Another widely studied testing strategy is ART [32]. Both the objective and approach of ART are different from AT. AT was developed to detect multiple faults. ART, on the other hand, was developed as an enhancement to RT with the objective of using fewer test cases to detect the first failure (referred to as the *F*-measure). ART is based on the intuition that when failure-causing inputs are clustered, selecting an input close to previously executed nonfailure-causing test cases will be less likely to detect a failure. Therefore, ART is proposed to have test cases evenly spread over the entire input domain. The test case is not selected randomly but rather selected by calculating the total distance from all previously selected test cases such that the total distance is maximized. For example, there are several test cases that have been executed without revealing any failure. Then we calculate the distances, using the Euclidean distance, between the unchosen test cases and the executed test cases. The test case that is furthest away from all executed test cases will be selected. Experimental results have shown that ART significantly outperforms traditional RT strategy. Since the inception of ART, many ART-based algorithms have been proposed. For example, mirror ART (MART) has been proposed by Kuo [33], which can be generally applied to improve the efficiency of ART algorithms based on the combination of "divide-and-conquer" and "heuristic" strategies. MART first divides the whole input domain into equal-sized disjoint subdomains. Then, one subdomain is chosen as the source domain while others as the mirror domains. Voronoi-ART has been proposed by Chen Merkel [27] to reduce the computational cost. The Voronoi-ART takes advantage of the properties of the Voronoi tessellation to achieve test case diversity, which decreases the computational cost to linear level. Ciupa *et al.* [34] have proposed an extension of ART for testing object-oriented applications (ARTOO) by defining the distance between objects. ARTOO have shown better performance than RT strategy. An automatic testing tool, named DART, has been proposed by Godefroid *et al.* [35] to enable automatic generation of the testing environment as well as test inputs. It can analyze the behavior of the SUT under RT and generate new test inputs to direct systematically, the execution along alternative program paths. Besides, some other improved ART algorithms, such as MART through dynamic partitioning [36], Lattice-based ART [37], quasi-random testing [38], randomized random testing, [39] and random border centroidal Voronoi tessellations [40] have also been proposed.

ART is designed to evenly distribute random test cases across the input domain to effectively detect the failure regions due to the observation that failures are usually clustered in the input domain. It is proposed as an enhancement of RT. However, the computational cost of ART is high, and its performance is usually unsatisfactory under high dimensions or nonnumeric environments. Many improved methods have been invested in ART to promote its effectiveness and reduce its algorithm complexity.

Both ART and DRT are based on the intuition that failure-causing test cases are clustered in the input domain. However, we do not include ART in our experiment for the following reasons. First, ART usually requires the numerical inputs, however, the test cases of subjects used in our experiment are hard to transfer to numerical ones. Second, DRT is to detect multiple faults by deploying as fewer test cases as possible. The objective of ART, on the other hand, is to use fewer test cases to detect the first failure. Indeed, the idea of ART can be combined into DRT to further improve the effectiveness when faced with numeric inputs. Then the comparison between DRT and ART can be investigated in our future work.

C. Dynamic Random Testing

DRT is in the context of software cybernetics which emphasizes the interplay of control science and software engineering. Some improved DRT strategies have been proposed to further improve the effectiveness, such as DRT-h [14], A-DRT [15], and O-DRT [16].

DRT-h strategy [14] uses the recent *h* results instead of the latest result to adjust the testing profile. In DRT-h, if there are faults detected in recent *h* times, the selection probability of subdomain with highest fault detection rate will increase from p_i to $p_i + \varepsilon$; otherwise, the selection probability of subdomain being selected most times will decrease from p_i to $p_i - \delta$. A-DRT [15] tunes the adjusting parameters during the testing process. The adjusting parameters have a great effect on the performance of DRT. Nevertheless, the adjusting parameters in the original DRT are constant, which might delay DRT's convergence to the optimal testing profile. Besides, Lv *et al.* [13] investigated the optimal values of adjusting parameters and provided a guideline for parameter settings to ensure that DRT can achieve a good effectiveness. A theoretical analysis was carried out to find out the the boundary values of adjusting parameters, in which DRT can obtain greater performance when ε/δ lies in the optimal interval, i.e., $[1/\theta_M - 1, 1/\theta_\Delta - 1]$, where θ denotes the fault

detection rate of the subdomains, $\theta_M = \{\theta_i | i-1, 2, ..., m\}$ and$\theta_\Delta = \{\theta_i | i-1, 2, ..., m, i \neq M\}$. Therefore, in A-DRT, the adjusting parameters ε and δ are modulated after the execution of the test cases to guarantee that the value of ε/δ is in the range of $[1/\theta_M - 1, 1/\theta_\Delta - 1]$. Further, since the theoretical foundation of the adjustment of fault detection rates in the optimization process is not explicit in A-DRT and its frequency of adjustment is fixed, O-DRT has been proposed [16], which changes the testing profile to a theoretically optimal one when the predefined criterion is satisfied. The theoretically optimal testing profile is calculated based on an optimization goal of both maximizing the overall fault detection rate and minimizing its variance. More specifically, an objective function, in which the benefit and the risk of using the current testing profile are taken into account, is constructed to quantify the performance of the testing profile. If the performance does not meet the predefined expectation, the testing profile will be adjusted immediately to optimize the objective function.

The purpose of DRT-h, A-DRT, and O-DRT is to enhance the fault detection effectiveness of the original DRT by using more testing results, tuning the adjusting parameters during the testing process, or utilizing optimization functions to adjust the testing profile. Some case studies have been reported, and the experimental results have shown that they can improve the effectiveness under certain circumstances. These improved DRT strategies provide potential directions of improvements on DRT; however, there still exist some problems in their works. For example, the advantage of A-DRT and O-DRT over the original DRT is in the case that the adjusting parameters ε and δ are set to be 0.05 at the beginning. The performance of DRT under other initial values are not investigated. Besides, the parameter ε is fixed and only δ changes during the testing process in their studies, which does not cover all possible value range. In this paper, the effectiveness and efficiency of DRT are evaluated by comparing with O-DRT and other testing strategies (shown in Section IV). Besides, an investigation on the effect of adjusting parameters is conducted in this paper (shown in Section V-A), in which both ε and δ are set at different values. Our experimental results can provide better data support for future improvement of DRT.

III. DRT Strategy and Analysis

In this section, the supplemented DRT strategy is introduced, and the factors that have an impact on DRT are analyzed.

A. DRT Strategy and Supplement

The principle of DRT is to update the testing profile dynamically during the testing process according to testing data collected online. Thus, the subdomains with higher fault detection rate are selected more easily. Suppose that test suite is divided into *m* subdomains $\{C_1, C_2, ..., C_m\}$, where C_i comprises k_i distinct test cases and the probability of C_i being selected is p_i. If C_i is selected, then the probability of each test case of C_i selected for execution is $1/k_i$. The executed test case may or may not detect faults. Then the testing results are used to adjust the testing profile dynamically. If a fault is detected by a test case selected from C_i, then p_i will increase to $p_i + \varepsilon$, and the selection probabilities of other subdomains will decrease $\varepsilon/(m-1)$; otherwise, p_i will decrease to $p_i - \delta$, and $\delta/(m-1)$ will be added on the selection probabilities of the other subdomains. The selection probabilities of all subdomains should be in the range of [0–1].

However, in previous studies, some of the experiment assumptions are inconsistent with real scenarios. First, the test suite keeps unchanged during the testing process, which means that the next test case is selected from the whole set of test cases including the executed test cases. Although it just needs to prepare a set of test cases at the beginning of the test, the algorithm is oversimplified because it may lead to the re-execution of the executed test cases and has a negative effect on the effectiveness of DRT. Therefore, in this paper, test cases are discarded from the test suite after being executed, so that the executed ones will not be selected in the following testing process. In this way, the effectiveness of DRT can be enhanced. Second, the fault detection process, from revealing a failure to detecting the faults, is not explained clearly. In a real testing process, failure is triggered by test cases if there appear abnormal outputs, errors, or warnings during the testing process. Then, the code review or fault diagnosis methods can be used to identify the faults; however, these methods are time-consuming and difficult to deploy in controlled experiments. Therefore, we seed faults in software and utilize the program instrumentation method to make the faults traceable. If the result from the execution of the test case deviates from expectations, we can know that there exist faults on the execution path of the test case, then the faults can be detected and located based on the outputs of the instrumentation.

The DRT algorithm is as follows.

Step 1: Partition the test suite into *m* subdomains $\{C_1, C_2, ..., C_m\}$ which consist of $k_1, k_2, ..., k_m$ distinct test cases; $k_1 + k_2 + \cdots + k_m = k$, where *k* is the total number of test cases.

Step 2: Initialize the testing profile $\{p_1, p_2, ..., p_m\}$ and adjusting parameters, i.e., ε and δ.

Step 3: Select a subdomain according to a given probability distribution $\{p_1, p_2, ..., p_m\}$, in which the probability of the ith subdomain C_i being selected is p_i. Suppose the lth subdomain C_l is selected, where $l \in [1, m]$.

Step 4: Select a test case from C_l in accordance with uniform distribution; that is, each distinct test case in the lth subdomain has an equal probability of $1/k_l$ being selected.

Step 5: Execute the selected test case and record the testing result (pass/fail).

Step 6: If a failure is triggered and there are new faults being detected by the test case selected from C_l, then

$$p_j = \begin{cases} p_j - \frac{\varepsilon}{m-1}, & p_j \geq \frac{\varepsilon}{m-1} \\ 0, & p_j < \frac{\varepsilon}{m-1} \end{cases} \quad \text{for } j \neq l, j = 1, 2, ..., m$$

$$p_l = 1 - \sum_{j \neq l} p_j. \tag{1}$$

Step 7: If no failure is triggered, or failure is triggered by the recorded faults, then

$$p_l = \begin{cases} p_l - \delta, & p_l \geq \delta \\ 0, & p_l < \delta \end{cases}$$

$$p_j = \begin{cases} p_j + \frac{\delta}{m-1}, & p_j \geq \delta \\ p_j + \frac{p_l}{m-1}, & p_j < \delta \end{cases} \text{ for } j \neq l, j = 1, 2, ...m. \quad (2)$$

Step 8: Record the faults being detected and remove the executed test case from the test suite.

Step 9: Check the subdomain being selected. If the subdomain is exhausted, then allocate its selection probability to other subdomains that possess test cases equally.

Step 10: Check the given testing stopping criterion. If it is not satisfied, then go to Step 3.

Step 11: Stop the software testing process.

As has been discussed, DRT can provide improvements over traditional testing strategies to some extent. A basic experiment is conducted in this paper to verify the effectiveness and efficiency of the DRT strategy. The fault detection effectiveness is measured in terms of the number of executed test cases to detect a given number of faults. The efficiency is measured by computational costs of testing strategies.

B. Analysis of Factors

Previous works always inspire further research. The adjusting parameters, initial profile, and test case classification are three critical factors in DRT and should be set before the testing process. However, they are still not investigated in-depth. Therefore, a series of extended experiments are conducted in this paper to investigate these factors.

1) The adjusting parameters might have a considerable impact on the performance of the DRT strategy, which has been preliminarily investigated in previous works [13], [15], [16]. Nevertheless, the analysis of adjusting parameters has not been investigated enough. On the one hand, the value of the parameter ε is fixed in their experiments groups and only δ changes from group to group, which does not cover the whole value range of the adjusting parameters. On the other hand, the real fault detection rates of subdomains are required before the testing, which is difficult to obtain in the real situation. Hence, the effects of the parameter variation on the performance of DRT still require further investigation. Both ε and δ should be set from 0 to 1 to cover the possible value range, and more combinations of the parameter values should be considered. The investigation of adjusting parameters is shown in Section V-A.
2) The initial profile is usually determined by the engineers according to previous experience or external information. However, because of the lack of pretest usage information in real projects, the initial profile is usually set as a uniform distribution profile. Although the feedback mechanism allows the DRT algorithm to gradually correct the bias of the testing profile during the testing process, the influence of the initial profile is still worthy of study. In this paper, three kinds of initial profiles, including the uniformly distributed profile, the proportionally distributed profile, and the fault detection rate distributed profile, are examined to verify the effect of initial profile. The experiments on the different initial profiles are illustrated in Section V-B.

The proportionally distributed profile means that the selection probability of subdomain *i* is equal to the ratio of the number of test cases in subdomain *i* to the total number of test cases in the entire test suite

$$p_i = \frac{k_i}{k}, \sum_{i=1}^{m} k_i = k, (i = 1, 2, ..., m) \quad (3)$$

where k_i denotes the number of test cases in the *i*th subdomain, and k denotes the total number of test cases in the entire test suite.

Although the feedback mechanism allows the DRT algorithm to gradually correct the bias of the testing profile during the testing process, the influence of the initial profile is still worthy of study. Therefore, another initial profile, named the fault detection rate distributed profile, is added in our experiments.

The fault detection rate distributed profile means that the selection probabilities of subdomains are in direct proportion to their real fault detection rates

$$p_i = \frac{\theta_i}{\sum_{j=1}^{m} \theta_j}, j = 1, 2, ..., m \quad (4)$$

where θ_i denotes the fault detection rate of subdomain *i,* namely $\theta_i = a_i/k_i$, in which k_i represents the number of test cases in subdomain *i* and a_i represents the number of test cases in subdomain *i* that can detect faults. It is important to note that the fault detection rate distributed profile is difficult to obtain in a real testing scenario. We use it as a comparative initial profile to investigate the performance of DRT under different types of initial profiles.

3) The test case classification is another key issue in DRT. The fault detection rates of subdomains might vary under different classification methods, which might have an effect on the testing process and the performance of DRT. The principle of DRT strategy is based on the Pareto rule, in which the discrepancies among the fault detection rates of subdomains are comparatively apparent. In this situation, the selection probabilities of subdomains with high fault detection rates will increase during the testing process in most cases, and the faults could be detected earlier and faster. Otherwise, if the discrepancies of fault detection rates among subdomains are minimal, the performance of DRT might be undesirable. Therefore, it is worth to investigate how the test case classification has an impact on the performance of DRT.

Three classification methods, i.e., the functional classification method, the combination classification method, and the random classification method are adopted in this paper to preliminary explore the effect of classification methods.

The functional classification method partitions the test suite into subdomains according to function criteria. The test cases partitioned into the same subdomain have similar

characteristics such as the same parameters, script files, or input text files. The functional classification method is also used in basic experiments to partition the test suite.

The combination classification method combines different settings of the functions when partitioning the test suite. Suppose that function *X* includes two settings (*x1* and *x2*) and function *Y* includes another two settings (*y1* and *y2*). The subdomains can be obtained by selecting test cases that cover different settings from *X* and *Y*, such as case 1 (*x1*, *y1*), case 2 (*x1*, *y2*), case 3 (*x2*, *y1*), and case 4 (*x2*, *y2*). For instance, if a test case could cover the setting *x1* and *y2*, which mean that it meets case 2, then it should be partitioned into the second class. Different classifications of test cases can be obtained through combinational classification method.

The random classification method partitions the test suite randomly. It is usually employed as a benchmark for classification methods.

It should be noted that one test case can only be partitioned into one subdomain, i.e., the classified subdomains are disjoint. If a test case can be classified to more than one subdomain, then it will be assigned to one of the subdomains randomly to avoid the overlapping among subdomains. The investigation on test case classification is shown in Section V-C.

IV. Basic Experiments

In the basic experiment, we investigate the effectiveness and efficiency of DRT to answer the first two fundamental questions of DRT.

1) To investigate whether DRT can achieve better fault detection effectiveness over other testing strategies in actual testing for the software programs of various scale and functionality, we conduct an experiment to compare the effectiveness of DRT and O-DRT with that of RT, RPT, and AT in terms of *T*-measure and APFD.
2) To investigate how much computational cost will DRT incur and whether it will noticeably affect the efficiency of DRT, we record time consumptions on test case selection process of all testing strategies and subjects. Then we discuss the efficiency of DRT by comparing their computational costs.

A. Subject Programs

The basic experiment is conducted by using five real software programs. Several previously released versions were obtained from the Software-artifact Infrastructure Repository (SIR) [41] together with their test suites and injected faults. These subjects are commonly used as UNIX utilities developed by GNU. Basic information concerning each subject program is shown in Table I.

Faults are seeded into these subject programs according to the bug history and the codes that are deleted from, inserted into, or modified between the versions. There are modified versions and test suites for each subject, except several versions without any faults, such as *Gzip v3* and *Grep v1*.

All specification-based test cases are written in the TSL language, which is a category-partition method for creating functional test suites [42]. For example, *Gzip* (GNU zip) is a compression utility designed as a replacement for compress. Compared with *compress*, *Gzip* is free from patented algorithms and exhibits superior compression performance. *Flex* is a lexical analyzer generator, and it serves as a tool for generating programs that perform pattern matching on the text. There are several manners in which to partition the test suite [43], and a common method is based on functionality, which is adopted in this paper. Take *Grep,* for example, the classification depends on the type of input file, in which pattern size, quoting, embedded blanks, embedded quotes are parametric functions, and the number of occurrence of patterns in the file or directory and the pattern occurrence on the target line are experimental functions. The 477 test cases are partitioned into 3 disjoint subdomains on the basis of the functionalities that they are intended to test, containing 37, 295, and 145 test cases, respectively. Regarding *Make*, the 795 test cases are partitioned into five disjoint subdomains, containing 27, 192, 192, 192, and 192 test cases, respectively. The 1061 test cases in *Bash* are partitioned into 7 subdomains, containing 13, 45, 499, 192, 188, 41, and 83 test cases, respectively.

TABLE I
Subject Programs

Subject Program	Size (LOC)	Versions	# of faults	# of test cases	# of subdomains
Gzip	5680	v1-v2-v4-v5	5-3-3-4	214	4
Flex	10459	v1-v2-v3-v4	16-13-8-11	525	5
Grep	10068	v2-v3-v4	2-7-3	477	3
Make	35545	v1-v2	4-5	795	5
Bash	59846	v2-v3-v4-v5-v6	3-6-4-4-9	1061	7

We choose the above subject programs out of the numerous candidates offered by the SIR because we intend to examine the effectiveness and efficiency of DRT strategy via experiments on subject programs with different properties, e.g., different scales, functionalities, and numbers of faults and test cases. Additionally, by using different versions of these subjects, we can expand the number of experiments, which provides more testing data support for comparison of testing strategies.

B. Testing Strategies and Settings

In this set of experiments, there are five testing strategies being conducted, including RT, RPT, AT, DRT, and O-DRT.

1) RT randomly selects test cases from the entire test suite, which is not partitioned. In this paper, the uniform distribution profile is used as the testing profile. RT provides a baseline for performance comparison.
2) RPT first selects a subdomain from the entire test suite, i.e., each subdomain has an equal probability of being selected. Next, a test case is randomly chosen for execution from the selected subdomain.
3) AT is included to investigate whether DRT can come close to or even outperform the AT strategy, which employs

more complex modeling of the test process and more sophisticated parameter estimation methods. We use AT-GA in this section, in which the GA is employed for parameter estimation. AT-GA first selects a test case randomly and then uses the GA to select the following test cases.

4) DRT updates the testing profile dynamically during the testing process according to testing data collected online. The algorithm is shown in Section III-A. And we set $\varepsilon = \delta = 0.05$ tentatively.
5) O-DRT is an improved DRT strategy that adjusts the testing profile to an optimal one once the predefined condition is achieved. In O-DRT, the Bayesian estimation method [44], [45] is adopted to estimate the fault detection rate θ of each subdomain during the testing process. We assume that the prior distribution of θ is *Beta (a, b)* and the posterior (after the execution of test cases) distribution of θ is *Beta(x + a, t-x + b)*, where *t* is the number of test cases being selected, *x* is the number of test cases detecting faults, and *a* and *b* are the prior parameters. Bayesian estimation of θ and its variance are given by

$$\hat{\theta} = \frac{(x+a)}{(t+a+b)}, \hat{V} = \frac{(x+a)(t-x+b)}{(t+a+b)^2(t+a+b+1)}. \tag{5}$$

Suppose that the test suite is partitioned into *m* subdomains $\{C_1, C_2, ..., C_m\}$, and $\hat{\theta}_i$ is the estimated fault detection rate of each subdomain. The prior parameters of subdomain C_i are denoted as a_i and b_i. The estimation of overall $\widehat{\theta}$ and its variance are calculated as in (4)

$$\hat{\theta} = \sum_{i=1}^{m} p_i \widehat{\theta}_i, \hat{V} = \sum_{i=1}^{m} {p_i}^2 \widehat{V}_i \tag{6}$$

where p_i is the selection probability of subdomain C_i.

Then the objective function is constructed based on the testing profile. The estimated value of θ_i and V_i should be normalized first, then the overall fault detection rate $\hat{\theta}'$ and its variance $\hat{V}'$ are calculated based on the normalized values

$$\widehat{\theta'_i} = \frac{\widehat{\theta}_i}{\widehat{m\theta}}, \widehat{V'_i} = \frac{\widehat{V}_i}{\widehat{mV}} \tag{7}$$

$$\hat{\theta}' = \sum_{i=1}^{m} p_i \widehat{\theta'_i}, \hat{V}' = \sum_{i=1}^{m} {p_i}^2 \widehat{V'_i} \tag{8}$$

where $\widehat{m\theta}$ and $\widehat{mV}$ are the estimated values of $\max(\theta_i)$ and $\max(V_i)$ $(i \in \{1, 2, \ldots, m\})$. And the objective function is calculated as follows:

$$f\left(\widehat{m\theta}, \widehat{mV}, p_1, \ldots, p_m, \widehat{\theta}_1, \ldots, \widehat{\theta_m}, \widehat{V_1}, \ldots, \widehat{V_m}\right) = \widehat{V}' - \widehat{\theta}'. \tag{9}$$

The optimization goal is to maximize $\widehat{V}'$ and minimize $\widehat{\theta}'$, i.e., minimize the function value. In O-DRT, the optimization is conducted every *q* $(q > 1)$ selections to avoid frequent adjustment. The algorithm of O-DRT is as follows.

Compared with DRT, O-DRT strategy initializes more parameters and adds a step to optimize the testing profile after the regular adjustment. It should be noted that one of the preconditions that requires the real fault detection rates of the test suite before testing is removed in our experiment since it cannot obtain in practice. In this experiment, the *q* is set to 5 to avoid frequent readjustments.

Algorithm O-DRT

1. Initialize testing profile, $\varepsilon = \delta = 0.05$, $a_i = b_i = 1$, $q = 5$.
2. Select a test case and execute it.
3. Record the testing results and adjust the profile according to formula (1) and (2).
4. Calculate the objective function *f* according to formula (9).
5. If $q \neq 0$, $q = q - 1$.
6. If $q = 0$, optimize *f* by minimizing its value, $q = 5$.
7. If all seeded faults are detected, then stop the testing process; otherwise, go to step 2.

Additionally, the test case classification of RPT, AT, DRT, and O-DRT are the same, namely, the test cases are partitioned according to functional criteria. Besides, the uniformly distributed profile is set as the initial profile for RPT, DRT, and O-DRT.

For all the testing strategies and subjects, we develop a testing platform to conduct the testing process automatically. The platform selects and executes test cases using the specific testing strategy. The platform uses an automated test "oracle" to detect failure, which means that whether a failure occurs is determined by comparing the faulty version with the correct version. The failure is triggered if the outputs of the two versions are different. Then, the execution trace for each test case can be obtained through the instrumentation tools.

DRT makes some assumptions that may not be completely consistent with the actual testing process. For example, if there is more than one fault on the trace of the failure-revealing test cases, then all the invoked faults are regarded as detected. However, this perfect debugging process does not conform to the actual testing process. The real debugging process belongs to the imperfect debugging, namely, the faults being detected might be not removed with certainty [46]. Besides, the debugging cost, which may occupy a lot of time in the real testing process, is also ignored in our controlled experiment. Indeed, these idealized settings can threaten the experimental validation, which will be further discussed in Section VI. Moreover, the selected test cases are executed and then discarded from the test suite.

The experiment is executed on a 64-bit Windows 7 PC, which is powered by a 3.20 GHz Intel Core i5-3470 Quad processor and has 8 GB RAM. The detailed experiment process is shown in Fig. 1. First, the adjusting parameters, initial profile, and partitioned test cases are set and prepared before testing. In basic experiment, the parameters ε and δ are set as 0.05, the uniformly distributed profile is set as the initial profile. Second, the experiment is conducted on five subjects with 18 versions. The test-stopping criterion is that all seeded faults are detected. The testing results, such as the number of executed test cases, and time consumed on test case selection are recorded. Third, O-DRT, RT, RPT, and AT strategies are conducted with a similar process. Because all these strategies are randomized algorithms,

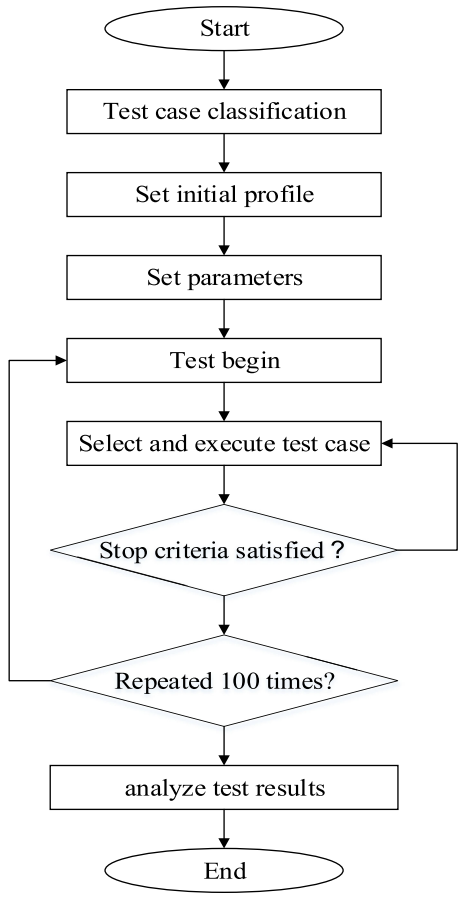

Fig. 1. Experiment process.

the number of trials for each scenario is empirically set to 100 to obtain satisfactory statistical performance of measurement. Finally, the effectiveness and efficiency of DRT and O-DRT are compared with those of RT, RPT, and AT.

C. Metrics of Experiment

To reflect both fault detection effectiveness as well as the efficiency of testing strategies, some metrics are introduced to evaluate the testing results.

1) The *T*-measure (the number of test cases executed to detect multiple faults) is one of the main metrics. The less number of executed test cases indicates higher effectiveness. Relevant metrics are shown below.

$\bar{T}$*:* We set *T* as the total number of test cases consumed to detect all seeded faults in experiments; each experiment is repeated 100 times to determine the average value of *T*, denoted as $\bar{T}$

$$\overline{T} = \frac{1}{100}\sum_{i=1}^{100} T_i. \tag{10}$$

D (Deviation): *D* is the standard variance of *T* over 100 trials

$$D = \sqrt{\frac{1}{99}\sum_{i=1}^{100}\left(T_i - \overline{T}\right)^2}. \tag{11}$$

CV: CV denotes the coefficients of variation over 100 trials, which can also be viewed as a normalized measure of dispersion

$$\mathrm{CV} = \frac{D}{\overline{T}} = \sqrt{\frac{1}{99}\sum_{i=1}^{100}\left(\frac{T_i}{\overline{T}} - 1\right)^2}. \tag{12}$$

Merely comparing the *T*-measure of two types of testing strategies is not sufficiently convincing to evaluate their performance. A hypothesis test should be introduced to verify whether there is a significant difference in fault detection capability between two testing strategies [47]. Since there is no or little information data of distribution before testing, and the hypothesis that the testing results obey the normal distribution are not supported by most testing data, there exists a certain risk in using a parametric test. Hence, a nonparametric test named the Mann-Whitney U test, which does not require the assumption of normal distributions, is adopted in this paper. It can be used to determine whether two independent samples were selected from populations having the same distribution. The formula of the Mann-Whitney U test is as follows:

$$U = NM + \frac{N(N+1)}{2} - \sum_{x_i}\mathrm{Rank}(x_i) \tag{13}$$

where *N* is the number of trails of the first strategy and *M* is the number of the trails of the second strategy. $\mathrm{Rank}(x_i)$ is the rank of the *i*th trails. Then, the *p-value* can be obtained by checking the forms. The *p-value* reflects the significance level between two groups of results, and $p < 0.05$ indicates that there is a significant difference between the fault detection effectiveness of these two strategies. In this paper, we use SPSS to calculate the *p-value* between two testing strategies. Some *p-values* are reported as 0.000 because only three significant figures are retained.

2) Whether testing strategies can detect faults faster is another assessment criterion. A good testing strategy can detect faults in the early stage. Relevant metrics are shown below.

APFD *(Average Percentage of Faults Detected)*: APFD measures the prioritization effectiveness in terms of the fault detection rate of a test suite [47], which is defined as follows:

$$\mathrm{APFD} = 1 - \frac{\mathrm{TF}_1 + \mathrm{TF}_2 + \cdots + \mathrm{TF}_a}{na} + \frac{1}{2n} \tag{14}$$

where *n* is the total number of test cases, a is the total number of detected faults, and $\mathrm{TF}_i\ (0 < i < \mathrm{n})$ denotes the smallest number of test cases in the sequence that needs to be executed to expose fault *i*. The value range of APFD is [0–1]. For any given test suite, n and a are fixed so that higher APFD values means that the average value of TF_i is lower and thus imply a higher fault-detection coverage rate. The presented APFD is the average value of 100 times experiments.

Effect size: The improvement magnitude of a strategy over another is measured by the *effect size* [48]. An unstandardized *effect size* measurement is used in this paper, the value of which is given by the following:

$$e = \frac{\mathrm{APFD}_1 - \mathrm{APFD}_2}{\sqrt{(V_1 + V_2)/2}} \tag{15}$$

where V_1 and V_2 represent the variance of APFD_1 and APFD_2, respectively. If $e = 0$, it means that the two strategies are equivalent. If $e > 0$, it expresses that the former is superior to the latter.

3) The *computational cost* of testing strategies is also essential, which reflects the complexity of testing strategy algorithm. The time consumption in selecting test cases indicates the efficiency of testing strategies. The lower computational cost means the more efficient strategy. The time consumptions in selecting test cases for RT, RPT, AT, and DRT strategies are all recorded in this paper.

D. Experimental Results and Analysis

The performance comparison in terms of $\bar{T}$, *D,* CV, APFD, APFD-*D,* and computational costs of testing strategies on five subjects with 18 versions are shown in Table II. The percentage reduction of $\bar{T}$ is calculated by dividing the difference between the corresponding data and that of DRT by the data of DRT. The average time is the average value of total execution time over 100 times.

Table III shows the *p-values* and *effect sizes* among testing strategies. The *p-values* are tabulated for the Mann-Whitney U test, and the *effect size* is calculated based on APFD and APFD-*D*. Besides, the comparison among testing strategies is shown in Table IV.

It can be seen from Tables II–IV that the numbers of test cases required by DRT and O-DRT are smaller than those required by RT and RPT in most cases. The advantage of DRT over RT in terms of $\bar{T}$ ranges from 7 to 711% (reduction of $\bar{T}$), especially in *Make* and *Bash*. The only exception is *Grep v3,* where RT uses fewer test cases than DRT. Compared with RPT, DRT shows a better performance, where DRT significant outperforms RPT in seven versions and has almost the same performance as RPT in the rest versions. O-DRT outperforms RT in terms of $\bar{T}$ in 14 versions and uses more test cases in *Gzip v2* and *Flex v4*, where the total number of executed test cases is small. And O-DRT has better performance than RPT in 12 versions, and inferior to RPT in four versions. However, compared with AT, both DRT and O-DRT show better performance in only four versions. The number of test cases used by DRT and O-DRT is much larger than AT in nine out of 18 versions. This means that the effectiveness of DRT and O-DRT is worse than that of AT. On the other hand, the difference in CV among RT, RPT, DRT, and O-DRT is slight, which reveals that DRT and O-DRT have at least the same robustness as RT and RPT. And the CV of DRT and O-DRT is much smaller than that of AT in most cases, which indicates that the robustness of DRT and O-DRT is better than AT. Moreover, when compared with DRT, O-DRT shows better performance in five versions and bad performance in four versions. The difference in the rest versions is minimal. The CV of DRT and that of O-DRT are almost equal in most cases, except several versions in *Flex*. The result indicates that O-DRT does not significant outperform DRT in this experiment.

It may also be noted from the experiment results that the values from APFD of DRT are greater than that of RT in 16 versions, RPT in the 16 versions, and AT in 12 versions. Similarly, the APFD of O-DRT is greater than RT in 15 versions, RPT in 14 versions, and AT in 11 versions. Meanwhile, the *effect size* of DRT and O-DRT versus RT, RPT, and AT in these versions is greater than 0, which means that DRT and O-DRT can find faults faster than other strategies in most instances. Besides, the APFD of DRT and O-DRT is close to each other, in which both strategies dominate nine versions.

DRT updates the testing profile during the testing process to increase the selection probabilities of subdomains with higher fault detection rates, which can increase the fault detection effectiveness. O-DRT constructs an objective function, and obtain an optimal testing profile in every few steps, which has a positive effect on effectiveness in some cases. However, it does not show the expected improvements mentioned in the previous work [16]. The reason might be that we remove the precondition used in original O-DRT, which requires the real fault detection rates before testing. And in this way, O-DRT becomes more realistic, and the fairness in the comparison among the testing strategies is guaranteed. Nevertheless, O-DRT is still a good testing strategy that can provide a potential direction of improvement for further study on DRT.

What's more, considering the computational cost in accordance with Table II, the average time taken by RT to select a test case ranges from 0.005 to 0.02 ms, and the corresponding time for RPT ranges from 0.04 to 0.06 ms, for DRT ranges from 0.04 to 0.07 ms, and for O-DRT range from 0.58 to 1.17 ms. However, AT takes more than hundreds even thousands of milliseconds to select a test case. The time consumed in DRT is a little more than RT and almost the same as RPT. The time used in O-DRT is about ten times than DRT since the optimization function behind O-DRT is time-consuming. However, the time consumption of both DRT and O-DRT are negligible compared with that of AT.

E. Summary

Compared with RT and RPT, DRT and O-DRT use fewer test cases to detect a fixed number of faults in most cases, while using more test cases than AT. And DRT has at least the same robustness as RT and RPT, and better than AT. Besides, DRT and O-DRT can detect faults faster than RT, RPT, and AT strategies in most instances, which is also an advantage of DRT. The computational cost of DRT is a little more than RT, and at the same order of magnitude as that of RPT. The computational cost of O-DRT is more than DRT. Nevertheless, both DRT and O-DRT use much less time to select test cases when compared with AT.

Furthermore, the comparison between DRT and other strategies are more comprehensive than that in previous works due to diversified metrics. And the time consumptions of all subjects, rather than only one subject, are recorded to avoid the instability of a single sample. We can conclude that DRT largely reduces the additional computation required by AT, and provides a performance improvement compared with that of RT and RPT.

TABLE II
BASIC EXPERIMENTAL RESULTS

Software subjects	Testing strategies	$\overline{T}$	Percentage reduction of $\overline{T}$ (vs DRT)	Percentage reduction of $\overline{T}$ (vs O-DRT)	*D*	*CV*	*APFD*	*APFD-D*	Average time(ms)	Time per test case(ms)
Gzip v1	RT	143.34	52.98%	113.40%	53.08	0.37	0.72	0.11	0.79	0.01
	RPT	110.91	18.37%	65.12%	35.96	0.32	0.78	0.08	4.59	0.04
	AT	38.8	-58.59%	-42.24%	11.09	0.29	0.92	0.03	201791.2	5200.80
	DRT	93.7	--	39.50%	32.50	0.35	0.81	0.07	4.1	0.04
	O-DRT	67.17	-28.31%	--	25.89	0.39	0.86	0.06	45.31	0.67
Gzip v2	RT	11.84	31.56%	-5.58%	10.75	0.91	0.98	0.02	0.2	0.02
	RPT	8.97	-0.33%	-28.47%	7.21	0.80	0.98	0.01	0.49	0.05
	AT	12.96	44.00%	3.35%	18.72	1.44	0.98	0.05	45267.12	3492.83
	DRT	9.01	--	-28.23%	7.12	0.79	0.98	0.01	0.36	0.04
	O-DRT	12.54	39.33%	--	9.30	0.74	0.98	0.02	14.22	1.13
Gzip v4	RT	147.24	48.98%	67.30%	48.56	0.33	0.64	0.14	1.03	0.01
	RPT	112.73	14.06%	28.09%	39.16	0.35	0.72	0.11	4.56	0.04
	AT	88.32	-10.63%	0.35%	83.01	0.94	0.81	0.16	378532.8	4285.92
	DRT	98.83	--	12.29%	36.44	0.37	0.75	0.10	4.58	0.05
	O-DRT	88.01	-10.95%	--	32.36	0.37	0.78	0.09	52.62	0.60
Gzip v5	RT	123.12	45.09%	52.04%	53.55	0.43	0.77	0.10	0.79	0.01
	RPT	89.68	5.68%	10.74%	40.75	0.45	0.83	0.08	3.7	0.04
	AT	84.58	-0.33%	4.45%	71.61	0.85	0.85	0.13	268234.3	3171.37
	DRT	84.86	--	4.79%	37.50	0.44	0.84	0.07	3.7	0.04
	O-DRT	80.98	-4.57%	--	37.95	0.47	0.85	0.07	47.99	0.59
Flex v1	RT	24.03	7.23%	-38.90%	19.31	0.80	0.99	0.00	0.19	0.01
	RPT	19.3	-13.88%	-50.93%	13.14	0.68	0.99	0.00	1.16	0.06
	AT	23.84	6.38%	-39.38%	31.03	1.30	0.99	0.01	96123.1	4032.01
	DRT	22.41	--	-43.02%	15.46	0.69	0.99	0.00	1.32	0.06
	O-DRT	39.33	75.50%	--	41.40	1.05	0.99	0.01	33.93	0.86
Flex v2	RT	292.11	300.86%	151.67%	123.56	0.42	0.93	0.03	2.19	0.01
	RPT	86.05	18.09%	-25.86%	28.47	0.33	0.97	0.01	5.14	0.06
	AT	37.7	-48.26%	-67.52%	49.45	1.31	0.99	0.01	112534.2	2984.99
	DRT	72.87	--	-37.22%	27.33	0.38	0.98	0.01	3.53	0.05
	O-DRT	116.07	59.28%	--	56.73	0.49	0.96	0.02	91.87	0.79
Flexv3	RT	360.07	410.52%	657.40%	101.09	0.28	0.80	0.06	2.63	0.01
	RPT	88.76	25.85%	86.71%	31.29	0.35	0.95	0.02	4.16	0.05
	AT	79.8	13.14%	67.86%	89.43	1.12	0.95	0.08	360213.8	4513.96
	DRT	70.53	--	48.36%	26.66	0.38	0.95	0.02	4.23	0.06
	O-DRT	47.54	-32.60%	--	29.19	0.61	0.96	0.03	45.82	0.96
Flex v4	RT	7.57	14.18%	-12.59%	5.26	0.69	1.00	0.00	0.19	0.03
	RPT	8.25	24.43%	-4.73%	6.16	0.75	1.00	0.00	0.4	0.05
	AT	6.6	-0.45%	-23.79%	3.10	0.47	1.00	0.00	4116.2	623.67
	DRT	6.63	--	-23.44%	4.42	0.67	1.00	0.00	0.47	0.07
	O-DRT	8.66	30.62%	--	5.34	0.62	0.99	0.00	10.02	1.16
Grep v2	RT	159.84	308.59%	337.68%	116.92	0.73	0.81	0.13	1.65	0.01
	RPT	42.34	8.23%	15.94%	27.51	0.65	0.95	0.03	1.78	0.04
	AT	18.1	-53.73%	-50.44%	11.69	0.65	0.98	0.01	12968.8	716.51
	DRT	39.12	--	7.12%	24.82	0.63	0.96	0.03	1.74	0.04
	O-DRT	36.52	-6.65%	--	24.32	0.67	0.96	0.03	42.59	1.17
Grep v3	RT	42.85	-21.68%	-27.03%	38.86	0.91	0.98	0.01	0.46	0.01
	RPT	61.34	12.12%	4.46%	45.94	0.75	0.98	0.01	2.91	0.05
	AT	113	106.54%	92.44%	200.85	1.78	0.96	0.06	365740.5	3236.64
	DRT	54.71	--	-6.83%	43.46	0.79	0.98	0.01	2.72	0.05
	O-DRT	58.72	7.33%	--	41.67	0.71	0.98	0.01	39.73	0.68

TABLE II
CONTINUED

Grep v4	RT	145.71	169.09%	159.22%	83.56	0.57	0.86	0.08	1.19	0.01
	RPT	57.08	5.41%	1.55%	27.12	0.48	0.94	0.03	2.47	0.04
	AT	71.5	32.04%	27.20%	82.70	1.16	0.94	0.06	161372.6	2256.96
	DRT	54.15	--	-3.66%	23.57	0.44	0.94	0.02	2.78	0.05
	O-DRT	56.21	3.80%	--	27.05	0.48	0.94	0.03	38.25	0.68
Make v1	RT	233.62	487.13%	624.85%	144.90	0.62	0.90	0.05	1.95	0.01
	RPT	45.79	15.08%	42.07%	29.35	0.64	0.98	0.01	2.05	0.04
	AT	44.5	11.84%	38.07%	59.12	1.33	0.98	0.04	369306	8299.01
	DRT	39.79	--	23.46%	22.13	0.56	0.98	0.01	2.01	0.05
	O-DRT	32.23	-19.00%	--	17.82	0.55	0.98	0.01	27.12	0.84
Make v2	RT	286.55	625.81%	695.75%	163.47	0.57	0.86	0.08	2.51	0.01
	RPT	53.25	34.88%	47.88%	29.31	0.55	0.97	0.02	2.48	0.05
	AT	45.2	14.49%	25.52%	62.70	1.39	0.98	0.03	154750.1	3423.67
	DRT	39.48	--	9.64%	22.56	0.57	0.98	0.01	2.06	0.05
	O-DRT	36.01	-8.79%	--	18.37	0.51	0.98	0.01	37.16	1.03
Bash v2	RT	309.51	771.86%	683.37%	201.47	0.65	0.80	0.13	2.79	0.01
	RPT	42.34	19.27%	7.16%	29.37	0.69	0.96	0.03	2.22	0.05
	AT	34.5	-2.82%	-12.68%	66.35	1.92	0.81	0.08	32671.42	947.00
	DRT	35.5	--	-10.15%	23.11	0.65	0.96	0.02	2.16	0.06
	O-DRT	39.51	11.30%	--	22.71	0.57	0.97	0.02	31.11	0.79
Bash v3	RT	258.44	649.97%	776.66%	144.77	0.56	0.92	0.03	2.69	0.01
	RPT	39.67	15.12%	34.57%	28.27	0.71	0.98	0.01	2.7	0.07
	AT	18.88	-45.21%	-35.96%	24.97	1.32	0.81	0.08	32562.7	1724.72
	DRT	34.46	--	16.89%	21.60	0.63	0.98	0.01	2.47	0.07
	O-DRT	29.48	-14.45%	--	23.65	0.80	0.98	0.01	26.13	0.89
Bash v4	RT	291.03	636.23%	567.81%	233.30	0.80	0.82	0.13	3.01	0.01
	RPT	43.46	9.94%	-0.28%	25.67	0.59	0.97	0.02	3.01	0.07
	AT	16.5	-58.26%	-62.14%	32.52	1.97	0.81	0.08	43623.1	2643.82
	DRT	39.53	--	-9.29%	25.45	0.64	0.97	0.02	2.85	0.07
	O-DRT	43.58	10.25%	--	25.85	0.59	0.97	0.02	36.31	0.83
Bash v5	RT	300.72	662.47%	749.97%	233.08	0.78	0.79	0.15	3.69	0.01
	RPT	40.77	3.37%	15.23%	27.56	0.68	0.96	0.03	2.63	0.06
	AT	16.74	-57.56%	-52.69%	15.06	0.90	0.81	0.08	33621.13	2008.43
	DRT	39.44	--	11.48%	22.79	0.58	0.96	0.02	3.01	0.08
	O-DRT	35.38	-10.29%	--	21.95	0.62	0.97	0.02	30.93	0.87
Bash v6	RT	349.91	435.60%	361.99%	250.45	0.72	0.67	0.24	4.16	0.01
	RPT	87.69	34.23%	15.78%	46.33	0.53	0.92	0.04	5.86	0.07
	AT	28.78	-55.95%	-62.00%	42.08	1.46	0.81	0.08	82634	2871.23
	DRT	65.33	--	-13.74%	40.29	0.62	0.94	0.04	5.33	0.08
	O-DRT	75.74	15.93%	--	41.21	0.54	0.93	0.04	57.12	0.75

V. FACTOR ANALYSIS OF DRT

The investigation in this section is to answer the third fundamental questions of DRT. As discussed before, there are three major factors in DRT strategy, i.e., adjusting parameters, initial profile, and test case classification. In this section, three extended experiments are conducted to explore the effect of such factors on the performance of DRT.

A. *Effect of Adjusting Parameters (ε and δ)*

The adjusting parameters can be set to any value in the range of 0 to 1. Intuitively, the value of adjusting parameters ε and δ should not be too high or too low. If the values are too high, the testing profile will fluctuate drastically; otherwise, it will take a long time for the testing profile to achieve an optimal one. More specifically, consider an extreme case in which ε and δ

TABLE III
P-VALUE AND EFFECT SIZE

Subjects	*p-value*	RT	RPT	AT	DRT	O-DRT	*Effect size*	RT	RPT	AT	DRT	O-DRT
Gzip v1	DRT	0.000	0.000	0.000	--	0.000	DRT	30.0%	13.9%	-49.3%	--	-18.9%
	O-DRT	0.000	0.000	0.000	0.000	--	O-DRT	47.5%	29.6%	-29.3%	18.9%	--
Gzip v2	DRT	0.066	0.832	0.028	--	0.003	DRT	3.9%	-0.2%	2.3%	--	1.0%
	O-DRT	0.146	0.022	0.731	0.003	--	O-DRT	-0.9%	-1.0%	-0.6%	-1.0%	--
Gzip v4	DRT	0.000	0.003	0.032	--	0.011	DRT	32.4%	9.3%	-17.1%	--	-10.8%
	O-DRT	0.000	0.000	0.867	0.011	--	O-DRT	42.5%	20.1%	-7.7%	10.8%	--
Gzip v5	DRT	0.000	0.283	0.887	--	0.464	DRT	25.6%	4.8%	-1.7%	--	-3.5%
	O-DRT	0.000	0.132	0.453	0.464	--	O-DRT	27.1%	7.0%	-0.2%	3.5%	--
Flex v1	DRT	0.404	0.228	0.584	--	0.008	DRT	0.7%	-1.1%	4.0%	--	-1.0%
	O-DRT	0.004	0.000	0.009	0.008	--	O-DRT	1.0%	1.0%	0.6%	1.0%	--
Flex v2	DRT	0.000	0.001	0.000	--	0.000	DRT	36.3%	2.9%	-17.8%	--	17.0%
	O-DRT	0.000	0.004	0.000	0.000	--	O-DRT	19.2%	-8.6%	-25.4%	-17.0%	--
Flexv3	DRT	0.000	0.000	0.035	--	0.000	DRT	74.0%	6.9%	2.9%	--	-6.9%
	O-DRT	0.000	0.000	0.000	0.000	--	O-DRT	77.8%	6.9%	4.5%	6.9%	--
Flex v4	DRT	0.190	0.033	0.842	--	0.006	DRT	0.2%	1.3%	0.4%	--	12.4%
	O-DRT	0.276	0.647	0.007	0.006	--	O-DRT	-10.2%	12.5%	-9.6%	-12.4%	--
Grep v2	DRT	0.000	0.421	0.000	--	0.441	DRT	51.8%	2.5%	-15.0%	--	0.9%
	O-DRT	0.000	0.107	0.000	0.441	--	O-DRT	53.2%	5.1%	-16.1%	-0.9%	--
Grep v3	DRT	0.026	0.149	0.000	--	0.417	DRT	-5.0%	1.5%	8.3%	--	3.3%
	O-DRT	0.012	0.346	0.000	0.417	--	O-DRT	-3.9%	-2.4%	8.5%	-3.3%	--
Grep v4	DRT	0.000	0.612	0.021	--	0.579	DRT	37.7%	1.1%	3.0%	--	0.4%
	O-DRT	0.000	0.867	0.027	0.579	--	O-DRT	33.8%	-0.4%	-0.3%	-0.4%	--
Make v1	DRT	0.000	0.062	0.096	--	0.010	DRT	44.4%	1.3%	3.1%	--	-3.5%
	O-DRT	0.000	0.000	0.001	0.010	--	O-DRT	48.5%	3.5%	2.2%	3.5%	--
Make v2	DRT	0.000	0.000	0.012	--	0.461	DRT	57.6%	5.3%	2.5%	--	2.1%
	O-DRT	0.000	0.000	0.003	0.461	--	O-DRT	54.7%	6.0%	-1.6%	-2.1%	--
Bash v2	DRT	0.000	0.047	0.596	--	0.189	DRT	63.1%	4.0%	70.1%	--	-3.7%
	O-DRT	0.000	0.431	0.139	0.189	--	O-DRT	60.4%	3.3%	69.6%	3.7%	--
Bash v3	DRT	0.000	0.083	0.000	--	0.043	DRT	42.2%	2.3%	79.8%	--	-2.5%
	O-DRT	0.000	0.024	0.002	0.043	--	O-DRT	43.6%	2.5%	80.8%	2.5%	--
Bash v4	DRT	0.000	0.143	0.000	--	0.269	DRT	56.9%	2.8%	72.3%	--	1.2%
	O-DRT	0.000	0.863	0.000	0.269	--	O-DRT	54.7%	-1.5%	71.8%	-1.2%	--
Bash v5	DRT	0.000	0.768	0.000	--	0.162	DRT	61.7%	1.1%	68.8%	--	-5.4%
	O-DRT	0.000	0.269	0.000	0.162	--	O-DRT	60.9%	4.8%	70.5%	5.4%	--
Bash v6	DRT	0.000	0.002	0.000	--	0.073	DRT	72.5%	10.4%	53.1%	--	5.5%
	O-DRT	0.000	0.029	0.000	0.073	--	O-DRT	69.4%	4.6%	48.9%	-5.5%	--

are set to 0, which means that no adjustment will be made after the execution of a test case. In this case, DRT is simplified as RPT. If ε and δ are set to 1, the testing profile will be adjusted as $p_i = 0$ when a test case in C_i fails to detect faults, so that the next test case will definitely not be selected from C_i; otherwise, if a fault is detected by the test case in C_i, then the next test case will be selected from C_i only. Although it is unclear how the testing process would be affected by the adjusting parameters, it is reasonable to believe that there exist optimal values and suitable ranges of ε and δ where DRT can achieve desirable performance.

In previous works, a specific interval, i.e., $[1/\theta_M - 1, 1/\theta_\Delta - 1]$ in which ε/δ should lie was proposed to guarantee DRT's performance, and three pairs of parameters, proper, lower, and upper ones were examined corresponding to the ratio of ε to δ [13]. However, one of the adjusting parameter ε is fixed in this study, which does not cover all parameter settings. To further investigate the impact of ε and δ on the

TABLE IV
COMPARISON AMONG TESTING STRATEGIES

Comparison of testing strategies		Gzip (4versions)	Flex (4versions)	Grep (3versions)	Make (2versions)	Bash (5versions)	sum (18versions)
DRT vs RT	$\overline{T}_{DRT}<\overline{T}_{RT}$ && *p-value* < 0.05	3	2	2	2	5	14
	$\overline{T}_{DRT}>\overline{T}_{RT}$ && *p-value* < 0.05	0	0	1	0	0	1
	$APFD_{DRT}>APFD_{RT}$ (*effect size* > 0)	4	3	2	2	5	16
DRT vs RPT	$\overline{T}_{DRT}<\overline{T}_{RPT}$ && *p-value* < 0.05	2	3	0	1	2	8
	$\overline{T}_{DRT}>\overline{T}_{RPT}$ && *p-value* < 0.05	0	0	0	0	0	0
	$APFD_{DRT}>APFD_{RPT}$ (*effect size* > 0)	3	3	3	2	5	16
DRT vs AT	$\overline{T}_{DRT}<\overline{T}_{AT}$ && *p-value* < 0.05	1	1	1	1	0	4
	$\overline{T}_{DRT}>\overline{T}_{AT}$ && *p-value* < 0.05	2	1	2	0	4	9
	$APFD_{DRT}>APFD_{AT}$ (*effect size* > 0)	1	2	2	2	5	12
O-DRT vs RT	$\overline{T}_{O\text{-}DRT}<\overline{T}_{RT}$ && *p-value* < 0.05	3	2	2	2	5	14
	$\overline{T}_{O\text{-}DRT}>\overline{T}_{RT}$ && *p-value* < 0.05	0	1	1	0	0	2
	$APFD_{O\text{-}DRT}>APFD_{RT}$ (*effect size* > 0)	3	3	2	2	5	15
O-DRT vs RPT	$\overline{T}_{O\text{-}DRT}<\overline{T}_{RPT}$ && *p-value* < 0.05	2	1	0	2	2	7
	$\overline{T}_{O\text{-}DRT}>\overline{T}_{RPT}$ && *p-value* < 0.05	0	2	0	0	0	2
	$APFD_{O\text{-}DRT}>APFD_{RPT}$ (*effect size* > 0)	3	2	1	2	4	12
O-DRT vs AT	$\overline{T}_{O\text{-}DRT}<\overline{T}_{AT}$ && *p-value* < 0.05	0	1	2	1	0	4
	$\overline{T}_{O\text{-}DRT}>\overline{T}_{AT}$ && *p-value* < 0.05	1	3	1	1	4	9
	$APFD_{O\text{-}DRT}>APFD_{AT}$ (*effect size* > 0)	0	2	1	1	5	9
DRT vs O-DRT	$\overline{T}_{DRT}<\overline{T}_{O\text{-}DRT}$ && *p-value* < 0.05	1	3	0	0	0	4
	$\overline{T}_{DRT}>\overline{T}_{O\text{-}DRT}$ && *p-value* < 0.05	2	1	0	1	1	5
	$APFD_{DRT}>APFD_{O\text{-}DRT}$ (*effect size* > 0)	3	2	0	2	2	9

$\bar{T}_A < \bar{T}_B$ && *p-value* < 0.05 means that the number of test cases used by strategy A is significantly less than strategy B.

performance of DRT strategy, experiments with more specific combinations of ε and δ are conducted in this section. The ε and δ are separately set to different values from 0.005 to 1 at an interval of 0.005, which comprise $200 * 200 = 40\,000$ trials with different values of the parameters. The variation tendency of performance on DRT could be examined as the parameters change. The test is repeated 100 times for each trial. Besides, the rest of the experiment settings are the same for all scenarios. The initial profiles are set to uniformly distributed profile, and the functional classification method is used to partition the test suite, which is identical to that in basic experiment.

In this section, we choose five versions from each subject to conduct experiments to investigate the effectiveness of DRT under different settings of adjusting parameters. The tendency of effectiveness to ε/δ is also presented to find out if there exists an optimal interval for adjusting parameters. The experimental results are shown in Fig. 2. In the left graph of each subject, the *x*-axis is the value of the parameter δ, the *y*-axis is the value of the parameter ε, and the *z*-axis represents $\bar{T}$, namely the average number of executed test cases. In the right graphs, the *x*-axis is the ratio of ε with δ where the logarithmic coordinates are adopted since the value range of ε/δ is [1/200–200], and the y-axis represents $\bar{T}$.

As is shown in the left graphs of Fig. 2, DRT strategy has different performance under the different values of adjusting parameters. For each version, there exists an optimal area of ε and δ where $\bar{T}$ is lower than in other areas. The performance of DRT varying with the ratio of ε to δ can be seen from the right graphs. Taking *Gzip v1* for example, over 120 test cases are executed to detect the faults when ε/δ is around 1, while only 60 test cases are consumed when the value is 10. Correspondingly, there exist suitable values of ε/δ for each subject and the performance of DRT shows a "V" curve as ε/δ increases from 10^{-2} to 10^2, which implies that there might exist an optimal value for the adjusting parameters which lead to maximal effectiveness. For instance, the optimal value of ε/δ in *Flex v2* is around 10^2, and in *Grep v3 is* around 1.

Besides, when ε to δ are close to 0, DRT exhibits slightly worse performance in comparison to nearby areas, which can be seen from the left graphs. The most likely reason is that, when ε to δ are tiny, there is little room for the adjustment of the testing profile after the execution of the test cases so that the changing rate of the testing profile is rather slow. The feedback mechanism behind DRT requires a long period to work, which makes the effectiveness undesirable.

Compared with previous studies, the relationship between the adjusting parameters and the real fault detection rate can be further investigated. We execute all test cases in the test suite to determine which test cases can detect faults and provide the real fault detection rates of subdomains based the results. We find that when ε is fixed to 0.05 and the ratio of adjusting parameters is in the range of the limitation $[1/\theta_M - 1, 1/\theta_\Delta - 1]$, the DRT method will achieve better performance. However, there are other optimal areas in which the parameter ε is set to other values. This means that the theoretical analysis in previous works is correct; however, it should be extended to allow changing both adjusting parameters.

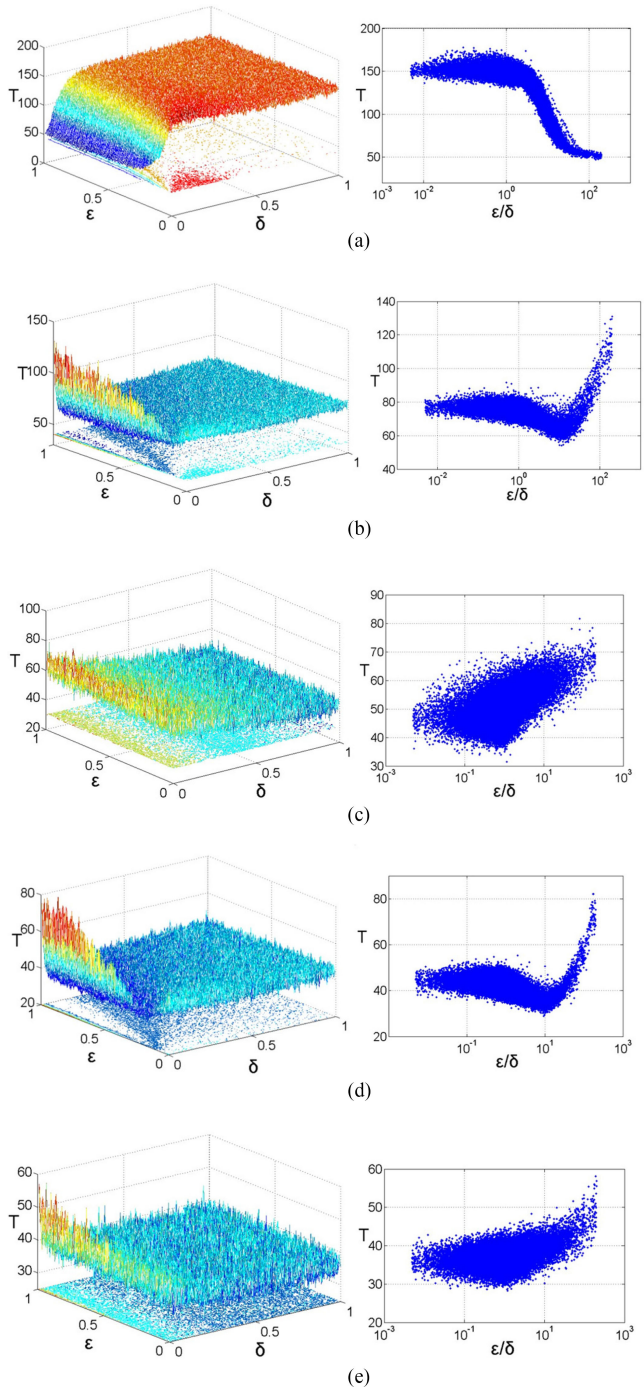

Fig. 2. Effect of adjusting parameters ε and δ on DRT (a) Gzip v1, (b) Flex v2, (c) Grep v3, (d) Make v2, and (e) Bash v2.

In conclusion, the performance of DRT is significantly affected by the adjusting parameters. DRT strategy with proper adjusting parameter settings could achieve better effectiveness when the other conditions are constant. Additionally, there exists an optimal area of ε and δ in each subject and the performance of DRT exhibits a "V" curve as ε/δ increases. However, because the optimal value may have relations with the fault detection rates of subdomains (which are difficult to obtain before testing), it is difficult to set the ideal adjusting parameters before testing. By comparing the testing results with previous work, we find that the theoretical analysis proposed before has some use; however, it still needs to be improved. The testing results obtained in this paper can reflect the effect of adjusting parameters from a more comprehensive point of view. Additionally, we should develop a proper method to change both adjusting parameters at the same time such that the effectiveness can be further improved.

TABLE V
EXAMPLE OF INITIAL PROFILE SETTING

Subdomains	# of test cases	# of test cases that can detect faults	UP	PP	FDP
1	51	19	25%	23.83%	62.15%
2	79	12	25%	36.92%	25.34%
3	80	6	25%	37.38%	12.51%
4	4	0	25%	1.87%	0

B. *Effect of Initial Profiles*

The initial profile is another major factor in DRT strategy. The initial profile can be set to any distributions before testing, and a uniformly distributed initial profile is an intuitive setting when we have little knowledge of the projects, although this might not be appropriate for all scenarios. Therefore, some other initial profiles, such as proportionally distributed profile and fault detection rate distributed profile, are adopted in this section. It should be noted that since the testing results of all test cases can be obtained in our controlled experiments, we can know whether test cases can detect faults. Hence, in fault detection rate distributed profile, the selection probabilities of subdomains are in direct proportion to their fault detection rates. It is intuitively regarded as a preferable initial profile.

Take *Gzip v1* for example, the test suite containing 214 test cases are partitioned into four subdomains through functional classification method. The total number of test cases and the number of the test cases that can detect faults in each subdomain are shown in Table V. More specifically, when using the uniformly distributed profile as the initial profile, the selection probabilities of all subdomains are $1/4 = 25\%$. When using proportionally distributed profile as the initial profile, the selection probability of subdomain 1 is $51/214 = 23.83\%$. When using the fault detection rate distributed profile as the initial profile, the selection probability of subdomain 1 is $\frac{19/51}{(19/51+12/79+6/80+0/4)} = 62.15\%$. UP represents the selection probabilities of subdomains in uniformly distributed profile, PP represents the selection probabilities in proportionally distributed profile, and FDP represents the selection probabilities in fault detection rate distributed profile.

Experimental results from DRT with the three initial profiles, namely DRT-UP, DRT-PP, and DRT-FDP, are compared to investigate the influence of the initial profile. Besides, the adjusting parameters are set as $\varepsilon = \delta = 0.05$ for all strategies. The experimental data for the comparison of testing strategies DRT-UP, DRT-PP, and DRT-FDP are shown in Table VI.

TABLE VI
EFFECT OF INITIAL PROFILE

Software Subjects	Testing strategies	$\overline{T}$	Reduction of $\overline{T}$ (with DRT-UP)	Reduction of $\overline{T}$ (with DRT-PP)
Gzip v1	DRT-UP	156.07	-	--
	DRT-PP	151	-3.25%	--
	DRT-FDP	143.21	-8.24%	-5.16%
Gzip v2	DRT-UP	13.56	-	--
	DRT-PP	13.94	2.80%	--
	DRT-FDP	11.31	-16.59%	-18.87%
Gzip v4	DRT-UP	162.56	-	--
	DRT-PP	161.26	-0.80%	--
	DRT-FDP	155.3	-4.47%	-3.70%
Gzip v5	DRT-UP	145.79	-	--
	DRT-PP	146.54	0.51%	--
	DRT-FDP	139.12	-4.58%	-5.06%
Flex v1	DRT-UP	30.46	-	--
	DRT-PP	25.61	-15.92%	--
	DRT-FDP	25.12	-17.53%	-1.91%
Flex v2	DRT-UP	76.03	-	--
	DRT-PP	91.17	19.91%	--
	DRT-FDP	82.11	8.00%	-9.94%
Flex v3	DRT-UP	81.37	-	--
	DRT-PP	99.46	22.23%	--
	DRT-FDP	78.21	-3.88%	-21.37%
Flex v4	DRT-UP	13.46	-	--
	DRT-PP	6.79	-49.55%	--
	DRT-FDP	10.13	-24.74%	49.19%
Grep v2	DRT-UP	34.47	-	--
	DRT-PP	48.06	39.43%	--
	DRT-FDP	31.23	-9.40%	-35.02%
Grep v3	DRT-UP	55.32	-	--
	DRT-PP	50.64	-8.46%	--
	DRT-FDP	55.22	-0.18%	9.04%
Grep v4	DRT-UP	59.92	-	--
	DRT-PP	66.58	11.11%	--
	DRT-FDP	54.15	-9.63%	-18.67%
Make v1	DRT-UP	38.73	-	--
	DRT-PP	51.46	32.87%	--
	DRT-FDP	35.13	-9.30%	-31.73%
Make v2	DRT-UP	47.84	-	--
	DRT-PP	58.67	22.64%	--
	DRT-FDP	47.23	-1.28%	-19.50%
Bash v2	DRT-UP	39.49	-	--
	DRT-PP	51.46	30.31%	--
	DRT-FDP	38.12	-3.47%	-25.92%
Bash v3	DRT-UP	36.61	-	--
	DRT-PP	50.84	38.87%	--
	DRT-FDP	33.21	-9.29%	-34.68%
Bash v4	DRT-UP	40.75	-	--
	DRT-PP	46.97	15.26%	--
	DRT-FDP	35.21	-13.60%	-25.04%
Bash v5	DRT-UP	42.77	-	--
	DRT-PP	58.7	37.25%	--
	DRT-FDP	39.32	-8.07%	-33.02%
Bash v6	DRT-UP	73.96	-	--
	DRT-PP	77.97	5.42%	--
	DRT-FDP	68.32	-7.63%	-12.38%

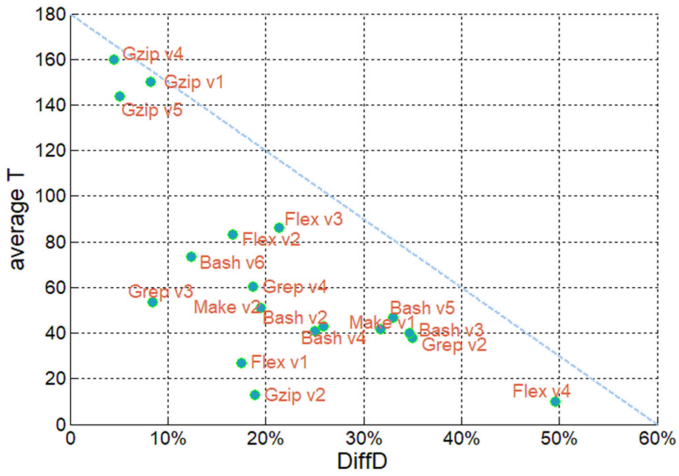

Fig. 3. Relationship between the difference degree and the number of executed test cases.

As shown in Table VI, DRT-UP has better performance than DRT-PP in terms of *T*, *D,* and CV in 13 out of 18 versions and worse than DRT-PP in the rest five versions. Compared with DRT-FDP, DRT-UP shows better performance in only one version (*Flex v2*), and DRT-PP has better performance in two versions (*Flex v4 and Grep v3*), which means that DRT with fault detection rate distributed profile can achieve higher fault detection effectiveness in most cases. This phenomenon conforms to the purpose of DRT, which aims at improving the selection probabilities with higher fault detection rates. If the initial profile is close to the "optimal one" (the uniformly distributed profile and proportionally distributed profile are usually far away from the "optimal one"), then DRT can achieve better effectiveness.

On the other hand, the differences among DRT-UP, DRT-PP, and DRT-FDP are not significant in most cases, i.e., the percentage reductions $\overline{T}$ shown in Table VI are not large. We can analyze the effect of the initial profile from another perspective, namely the relations between executed test cases and the difference among the three initial profiles. We denote IP as the set of $\overline{T}_{\mathrm{DRT}-UP}, \overline{T}_{\mathrm{DRT}-\mathrm{PP}}, \overline{T}_{\mathrm{DRT}-\mathrm{PP}}$, and the difference degree (Diff*D*) is calculated as follows:

$$\mathrm{Diff}D = \frac{\max(IP) - \min(IP)}{\max(IP)} \tag{16}$$

The relationship between the difference of initial profiles and the average executed test cases is shown Fig. 3, where the *x*-axis is the different degrees among DRT-UP, DRT-PP, and DRT-FDP; and *y*-axis represents the average executed test cases (*Average T*) used by DRT-UP, DRT-PP, and DRT-FDP.

As can be seen from Fig. 3, the difference among the effectiveness of DRT-UP, DRT-PP, and DRT-FDP are relatively small in most cases. The Diff*D* mainly distributed within the range from 10 to 35%. (As a comparison, the difference between DRT and other strategies are commonly larger than 40% even 60%). Besides, the main characteristic of the relationship

between Diff*D* and *Average T* is that almost all the points in the figure are located in the lower triangle of the first quadrant. It means that the Diff*D* could be relatively great when the number of executed test cases is relatively small, and it would be slight when the number of executed test cases is large. For example, the Diff*D* is obvious with respect to *Flex v4* (49.55%) and *Grep v2* (39.43%) where the number of executed test cases in *Flex v4* and *Grep v2* is quite small, and the Diff*D* is slight in *Gzip v4* (4.47%) and *Gzip v5* (5.06%), where the numbers of executed test cases are quite large. The possible reason is that the feedback mechanism of DRT automatically adjusts the bias of initial profile to best fit the actual fault detection rates of subdomains, but it needs a certain amount of time to achieve the optimal testing profile.

In conclusion, the initial profile has a certain influence on DRT. A good initial profile can improve fault detection effectiveness in most cases. However, the effect of the initial profile on effectiveness become less when the number of executed test cases is large.

C. Effect of Test Case Classification

Another issue examined in the extended experiments is the effect of the test case classification. The test case classification determines the fault detection rates of subdomains, although we cannot know it at the beginning of testing. In the basic experiment, the function classification method is used to partition the test suite. In this section, some other classification methods, such as the combination classification method and the random classification method are adopted to explore what effect that test case classification may have on the performance of DRT.

The functional classification method partitions the test suite into subdomains according to the functions that the test cases may cover. The combination classification method is combining the different settings of all the functions when classifying the test cases. The random classification method partitions the test suite into subdomains randomly, which is regarded as a benchmark in this section.

For example, all test cases in *Grep* have 14 parameters (which represent the functions they will test, and the value of these parameters can be set to 1 or 0) except for 37 test cases with single and error (which have no parameters). Usually, the test cases with single and error are classified into one subdomain since they are different from others. Then the rest of the test cases can be classified based on the value of parameters. Functional classification method partitions the test cases based on the value of several parameters, e.g., we partition the test cases based on the first and the second parameters. Since the first parameter (refer to as *Regexp Selection* and *Interpretation*) is set to 1 in all test cases, the second parameter (refer to as *Quoting*) are set to 1 in 295 test cases and 0 in the rest 145 test cases, we classify the 477 test cases into three disjoint subdomains, containing 37, 295, and 145 test cases, respectively. The combination classification method is used in the extended experiment to obtain new classification, e.g., we use the second and the third parameters (refer to as *Regular Expression Concatenated*) to partition the test cases into five subdomains, containing 37, 195, 50, 100, and 95 test cases, respectively. Besides, we use random classification method to generate matched groups, in which the number of subdomains equals that of the function classification method and the combination classification method.

There are four groups of experiments for each subject program. More specifically, the test suite is partitioned with functional classification method in the first experimental group; the test suite in the second experimental group is partitioned with random classification method, the number of subdomains of which is same with the first group. The combination classification method is adopted in the third experimental group, and the test suite in the fourth experimental group is partitioned into the same number of subdomains with the third group by random classification method. Besides, the uniform distribution is adopted to construct the initial profiles, and the parameters ε and δ are set to 0.05 in all scenarios. Fig. 4 presents the experimental results for each version with different classification methods. The x-axis represents the groups, for instance, "1" represents the first group in which the test cases are partitioned through the functional classification method. The y-axis is $\bar{T}$ (number of executed test cases).

As can be seen from Fig. 4, in the experiment of *Make* (*v1, v2*), *Grep* (*v2, v4*), *Flex* (*v1, v2, v3*), and *Bash* (*v2, v3, v4, v5, v6*), the effectiveness of DRT with functional classification method and combination classification method has significant advantage over that with corresponding random classification methods. Take *Bash v2* for example, DRT with functional classification method employs an average of 64.43 test cases to detect the faults, and 40.36 test cases with combination classification method, while in the random classification method groups, 299.64 and 311.38 test cases are consumed on average to detect all the seeded faults. Besides, the functional classification method and the combination classification method have no obvious difference in most cases, except for several versions in *Gzip*. In *Gzip* (*v1, v3, v4*), the effectiveness of DRT with functional classification method (the first experimental group) and that with corresponding random classification method (the second experimental group) is close.

According to the experimental results, we can conclude that the test case classification has a significant effect on the performance of DRT, and the functional classification method and combination classification method show better performance relative to the random classification method in most cases. The possible reason may stem from the Pareto rule, where the failure-causing test cases are usually clustered in the input domain. And the subdomains obtained by functional classification method and combination classification method may lead to an uneven distribution of fault detection rates. It is easier for DRT to select test cases from subdomains with higher fault detection rates in such cases. In contrast, it will require a higher computational cost for DRT to adjust the testing profile to an optimal one if the classification is undesirable. As a comparison, the fault detection rates of subdomains through the random classification are more likely to be evenly distributed since the classification process and selection process are both random. (This even distribution can be fully reflected through 100 trials.) There are minor differences when DRT selects test cases from these subdomains.

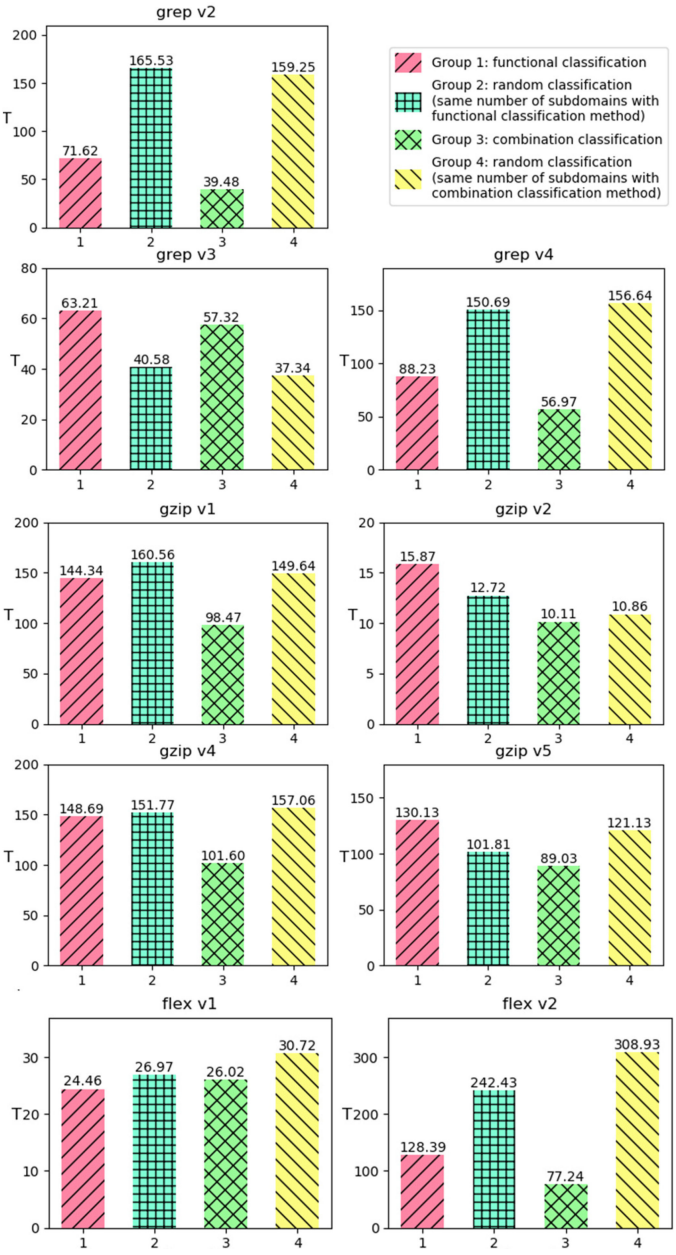

Fig. 4. Effect of test case classification.

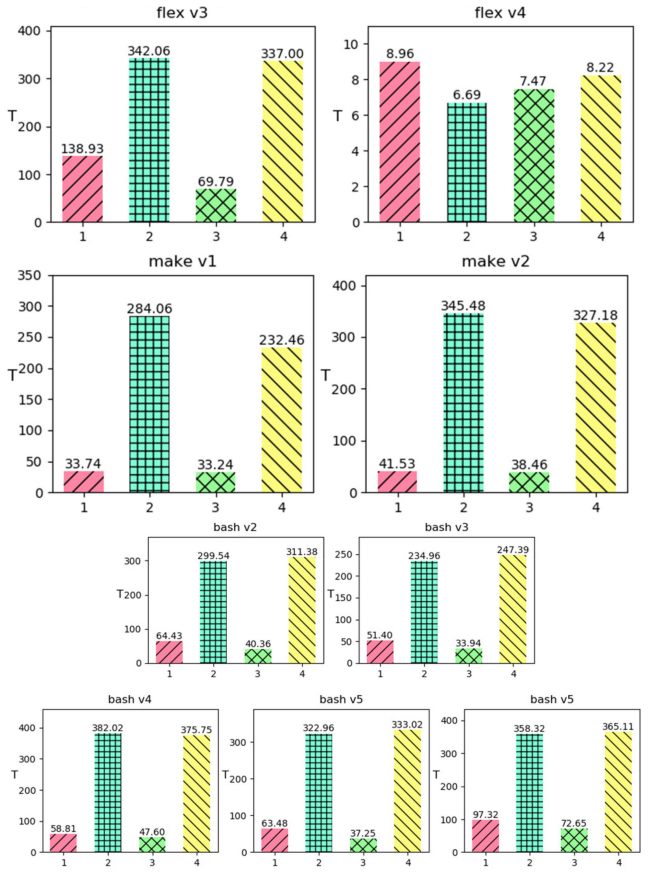

Fig. 4. Continued.

In conclusion, the test case classification has a significant influence on the performance of DRT. The effectiveness of DRT is affected by the quality of test case classification directly. A good classification should have an uneven distribution of fault detection rate among subdomains. Therefore, the functional classification and combination classification methods are recommended when applying DRT in practice.

D. Summary

The influence of the adjusting parameters ε and δ on the performance of DRT is quite remarkable. There exist an optimal area and a suitable ratio of ε to δ for each subject software. The adjusting parameters should not be too small, otherwise, DRT will degenerate into RPT and the performance is undesirable. Adjusting parameters are the unique property of DRT, which set DRT apart from traditional testing strategies and bring feedback mechanism to DRT. Although the fault detection rate of test cases are hard to obtain before testing, it is a possible way to dynamically adjust the parameters ε and δ during the testing process to achieve the optimal values and improve the effectiveness of DRT.

The initial profile may have a certain effect on the effectiveness of DRT, however, the effect should not be exaggerated. From experimentation results, the difference in performance among various initial profiles has a downward tendency when the number of executed test cases increases. It would be easy to deduce that the difference among other initial profiles will not be huge due to the feedback mechanisms of DRT. We can also conclude that DRT is robust to different initial profiles.

Test case classification also has noticeably influence on DRT, the performance of DRT would be better when the classification has an uneven distribution of fault detection rates. And when applying DRT in practice, the functional classification method and the combination classification method are the better methods to partition the test cases.

VI. Threats to the Validity

Like any case studies on software testing strategies, this study inevitably faces the following potential threats to validity.

First, the experiments in this paper are conducted with a limited number of subject programs, test suites, and faults, we cannot claim that the conclusions drawn from the experimental data could apply to elsewhere. This is the limitation of case studies in empirical software engineering in general. Nevertheless, the subject programs used in our experiments are also used in many other practical studies to compare the performance of different testing strategies. These subject programs are downloaded from the SIR, which have different origins with varying numbers of LOC, faults, and classifications of the test suite. And the faults seeded in the subjects are the real ones according to the bug reports, which could reflect the distribution of faults in real situations. Besides, the major motivation of this paper is not to provide a universal solution for every testing engineer but to introduce a new way of thinking when they look for alternative testing methods to improve or supplement RT and RPT techniques. The general principle of DRT can be applied in many other forms and tailored for different purposes. We look forward to future contributions to the application and evolution of this new technique.

Second, threats to validity in this paper may stem from how to define the performance of testing strategies and how the experiments can be designed without bias. There is always some bias in the results due to the randomness of empirical experiments. Therefore, some measures should be employed in our experiments to assess the testing strategies and decrease the bias of the testing results. There are many metrics that can be used to evaluate the effectiveness and efficiency of testing strategies. For example, the *T*-measure and APFD, which are used as metrics in this paper, can measure the fault detection effectiveness from different perspectives. The purpose of the *T*-measure is to determine whether testing strategies can use as few as possible testing resources to detect a given number of faults, and the purpose of APFD is to figure out whether testing strategies can detect faults faster. Since all the testing strategies are randomized, we adopt the *p-value* and the *effect size* to perform statistical analysis of the testing results, which can provide statistical significance and confidence in the results. Additionally, there are some other metrics that are not used, e.g., the *F*-measure, whose purpose is to use as few as possible test cases to find the first failure during the testing process. It may not be suitable for DRT, since DRT will utilize the testing results, especially those test cases that detect faults, to adjust the testing profile to achieve better performance. Picking the same number of test cases for all strategies to observe how many faults can be detected is also a common and fair metric. However, how many test cases should be executed is not easy to estimate when we have little information about the subject programs. These metrics can be used in our future work, especially when we have more knowledge about the SUT. On the other hand, the execution time of test cases is recorded to evaluate the efficiency. DRT strategy focuses on both effectiveness and efficiency to pursue a balance such that the effectiveness can be improved without high computational cost.

Third, the adjusting parameter, ε and δ, are set to 0.05 for DRT strategy in most of the experiments reported in this paper. According to the investigation presented in Section V-A, this setting might not be optimal for all subject. Therefore, some of the conclusions we present might not hold in the case where different parameter settings are used. How to obtain the optimal adjusting parameters remains a crucial and imminent issue for future studies on DRT.

Moreover, the assumption behind the fault detection process might have an effect on the performance of testing strategies. In our controlled experiment, we assume that the debugging process is perfect debugging, in which all the faults on the failed execution path are regarded as detected. This may make the fault detection effectiveness higher than that in the real testing process. Besides, the debugging cost is not discussed in this paper. The debugging cost makes the testing time less than that in a real situation. However, it has limited influence on the comparison results among the testing strategies in terms of effectiveness and efficiency. The reason is that the main difference among the testing strategies lies in the selection of test cases, and the debugging costs for all testing strategies should be very close.

VII. Conclusion

The importance of software testing has long inspired the development of several good test strategies to improve their effectiveness [48]. DRT has been found to present an improvement over RPT by dynamically updating testing profiles using testing results during the testing process. Compared with RPT, DRT strategy can overcome the drawback of the fixed selection probabilities of the subdomains and improve testing performance by increasing the probability of selecting test cases from subdomains with higher fault detection rates.

Several investigations have been proposed to enrich the research on DRT. However, there still exist some deficiencies. The previous experiments on the performance of DRT are incomplete since there lacks sufficient metrics to measure the effectiveness and efficiency of DRT. Several experimental assumptions are inconsistent with real scenarios. And the investigations of factors underlying DRT such as adjusting parameters, initial profile, and test case classification are not deep enough. Therefore, a DRT strategy with supplements were proposed in this paper, and a preliminary study with five real subjects (18 versions) was reported to investigate the effectiveness and efficiency of DRT by deploying plenty of metrics. Besides, three extended experiments were conducted to investigate the effect of factors on the performance of DRT.

The study gives answers to the fundamental questions raised at the beginning of this paper

1) When compared with RT and RPT, DRT has higher effectiveness in most cases. DRT can not only use fewer test cases to detect a given number of faults but also can detect faults faster than RT and RPT. And the robustness of DRT is close to that of RT and RPT. When compared with AT, DRT is less effective while has lower fluctuation. Besides, the performance of O-DRT is close to DRT, which can be considered as a good testing strategy.
2) The computational cost of DRT is a little more than RT and at the same order of magnitude as that of RPT. And it is almost negligible compared with that of AT. It can be

said that the computational cost of DRT is an advantage when applying the DRT strategy in practice.

3) The performance of DRT is noticeably affected by the adjusting parameter settings, and test case classification is robust to different initial profiles. First, the adjusting parameters have significant effects on DRT. There exist optimal values of adjusting parameters for each subject, where DRT can achieve better effectiveness. And the performance of DRT shows a "V" curve as ε/δ increases. Second, the initial profile has a certain effect on the performance of DRT, but the importance of this should not be overstated. The difference is comparatively small, especially when the number of executed test cases is large. It can be said that DRT's effectiveness is robust to different initial profiles. Third, the effect of test case classification is significant. The functional classification and combination classification could generate classifications with a more uneven distribution of fault detection rates, which are more desirable than the random classification method does.

Many issues deserve further research. A significant future improvement of DRT is dynamically changing the adjusting parameters during the testing process. Based on the discussion in Section V-A, there exist optimal values of adjusting parameters, which could be updated dynamically as more information is gathered from SUT. The objective function adopted in O-DRT can also be a possible direction of improvement. Some adaptive mechanisms could be introduced to update the adjusting parameters according to the testing results so that the values of parameters can stay in the desirable range. Besides, the idea of DRT can also be employed in other fields such as in resource allocation strategies under cloud testing environment. In this way, the resource allocation strategies can have better dynamics adjust characteristic, and the effectiveness can be further improved.

Indeed, DRT has great potential in theoretical as well as practical terms. This strengthens the feasibility of the idea of software cybernetics that is supposed to explore the interplay between the software testing strategies and the control theory.

REFERENCES

[1] J. Lv, H. Hu, K. Y. Cai, and T. Y. Chen, "Adaptive and random partition software testing," *IEEE Trans. Syst., Man, Cybern.: Syst.*, vol. 44, no. 12, pp. 1649–1664, Dec. 2014.

[2] K. Y. Cai *et al.*, "Random testing with dynamically updated test profile," in *Proc. 20th Int. Symp. Softw. Rel. Eng.*, 2009, pp. 1–2.

[3] W. J. Gutjahr, "Partition testing vs. random testing: The influence of uncertainty," *IEEE Trans. Softw. Eng.*, vol. 25, no. 5, pp. 661–674, Sep./Oct. 1999.

[4] T. Y. Chen and Y. T. Yu, "On the relationship between partition and random testing," *IEEE Trans. Softw. Eng.*, vol. 20, no. 12, pp. 977–980, Dec. 1994.

[5] X. Bai, S. Lee, W. T. Tsai, and Y. Chen, "Ontology-based test modeling and partition testing of web services," in *Proc. IEEE Int. Conf. Web Services*, 2008, pp. 465–472.

[6] P. Bernardi, L. Bolzani Poehls, M. Grosso, and M. Sonza Reorda, "A hybrid approach for detection and correction of transient faults in SoCs," *IEEE Trans. Dependable Secure Comput.*, vol. 7, no. 4, pp. 439–445, Oct.–Dec. 2010.

[7] J. Zeng, H. Lei, and H. Pu, "Evaluating the effectiveness of random and partition testing by delivered reliability," in *Proc. Int. Conf. Embedded Softw. Syst.*, 2008, pp. 496–502.

[8] A. Arcuri and L. Briand, "Formal analysis of the probability of interaction fault detection using random testing," *IEEE Trans. Softw. Eng.*, vol. 38, no. 5, pp. 1088–1099, Sep./Oct. 2012.

[9] P. J. Boland, H. Singh, and B. Cukic, "Comparing partition and random testing via majorization and Schur functions," *IEEE Trans. Softw. Eng.*, vol. 29, no. 1, pp. 88–94, Jan. 2003.

[10] F. T. Chan, T. Y. Chen, I. K. Mak, and Y. T. Yu, "Proportional sampling strategy: Guidelines for software testing practitioners," *Inform. Softw. Technol.*, vol. 38, no. 12, pp. 775–782, 1996.

[11] T. Y. Chen and Y. T. Yu, "On the expected number of failures detected by subdomain testing and random testing," *IEEE Trans. Softw. Eng.*, vol. 22, no. 2, pp. 109–119, Feb. 1996.

[12] K. Y. Cai, B. Gu, H. Hu, and Y. C. Li, "Adaptive software testing with fixed-memory feedback," *J. Syst. Softw.*, vol. 80, no. 8, pp. 1328–1348, 2007.

[13] J. Lv, H. Hu, and K.Y. Cai, "A sufficient condition for parameters estimation in dynamic random testing," in *Proc. IEEE 35th Annu. Comput. Softw. Appl. Conf. Workshops*, 2011, pp. 19–24.

[14] L. Zhang, B.B. Yin, J. Lv, K. Y. Cai, S. S. Yau, and J. Yu, "A history-based dynamic random software testing," in *Proc. IEEE 38th Int. Comput. Softw. Appl. Conf. Workshops*, 2014, pp. 31–36.

[15] Z. Yang, B. B. Yin, J. Lv, K. Y. Cai, S. S. Yau, and J. Yu, "Dynamic random testing with parameter adjustment," in *Proc. IEEE 38th Int. Comput. Softw. Appl. Conf. Workshops*, 2014, pp. 37–42.

[16] Y. Li, B.B. Yin, J. Lv, and K.Y. Cai, "Approach for testing profile optimization in dynamic random testing," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, 2015, vol. 3, pp. 466–471.

[17] J. J. Marciniak, *Encyclopedia of Software Engineering*, New York, NY, USA: Wiley, 1994, vol. 2, pp. 873–877.

[18] P. Bourque and R. E. Fairley, *Guide to the Software Engineering Body of Knowledge (SWEBOK (R)): Version 3.0,* Piscataway, NJ, USA: IEEE Computer Society Press.

[19] A. Orso and G. Rothermel, "Software testing: A research travelogue (2000–2014)," in *Proc. Future Softw. Eng.*, 2014, pp. 117–132.

[20] D. Cotroneo, A. Lanzaro, and R. Natella, "Faultprog: Testing the accuracy of binary-level software fault injection," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 1, pp. 40–53, Jan.–Feb. 2016.

[21] A. Arcuri, M. Z. Iqbal, and L. Briand, "Random testing: Theoretical results and practical implications," *IEEE Trans. Softw. Eng.*, vol. 38, no. 2, pp. 258–277, Mar./Apr. 2012.

[22] D. Amalfitano, N. Amatucci, A. R. Fasolino, P. Tramontana, E. Kowalczyk, and A. M. Memon, "Exploiting the saturation effect in automatic random testing of Android applications," in *Proc. 2nd ACM Int. Conf. Mobile Softw. Eng. Syst.*, 2015, pp. 33–43.

[23] Z. Zheng and G. Xiao, "Evolution analysis of a UAV real-time operating system from a network perspective," *Chinese J. Aeronaut.*, vol. 32, pp. 176–185, 2018.

[24] P. G. Frankl and E. J. Weyuker, "A formal analysis of the fault-detecting ability of testing methods," *IEEE Trans. Softw. Eng.*, vol. 19, no. 3, pp. 202–213, Mar. 1993.

[25] P. G. Frankl and E. J. Weyuker. Provable improvements on branch testing," *IEEE Trans. Softw. Eng.*, vol. 19, no. 10, pp. 962–975, Oct. 1993.

[26] E. J. Weyuker and B. Jeng, "Analyzing partition testing strategies," *IEEE Trans. Softw. Eng.*, vol. 17, no. 7, pp. 703–711, Jul. 1991.

[27] T. Y. Chen and R. Merkel, "Efficient and effective random testing using the Voronoi diagram," in *Proc. Aust. Softw. Eng. Conf.*, 2006, pp. 6–299.

[28] J. W. Duran and S.C. Ntafos, "An evaluation of random testing," *IEEE Trans. Softw. Eng.*, vol. 10, no. 4, pp. 438–444, Jul. 1984.

[29] R. Hamlet and R. Taylor, "Partition testing does not inspire confidence," *IEEE Trans. Softw. Eng.*, vol. 16, no. 12, pp. 1402–1411, Dec. 1990.

[30] K. Y. Cai, "Optimal software testing and adaptive software testing in the context of software cybernetics," *Inform. Softw. Technol.*, vol. 44, no. 14, pp. 841–855, 2002.

[31] J. Lv, B. B. Yin, and K. Y. Cai, "On the asymptotic behavior of adaptive testing strategy for software reliability assessment," *IEEE Trans. Softw. Eng.*, vol. 40, no. 4, pp. 396–412, Apr. 2014.

[32] T. Y. Chen, T. H. Tse, and Y.T. Yu, "Proportional sampling strategy: A compendium and some insights," *J. Syst. Softw.*, vol. 58, no. 1, pp. 65–81, 2001.

[33] F. C. Kuo, "An indepth study of mirror adaptive random testing," in *Proc. 9th Int. Conf. Qual. Softw.*, 2009, pp. 51–58.

[34] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer, "Object distance and its application to adaptive random testing of object-oriented programs," in *Proc. 1st Int. Workshop Random Testing*, 2006, pp. 55–63.

[35] P. Godefroid, N. Klarlund, and K. Sen, "DART: Directed automated random testing," in *Proc. ACM SIGPLAN Conf. Program. Lang. Design Implementation*, 2005, pp. 213–223.
[36] R. Huang, H. Liu, X. Xie, and J. Chen, "Enhancing mirror adaptive random testing through dynamic partitioning," *Inform. Softw. Technol.*, vol. 67, pp. 13–29, 2015.
[37] J. Mayer, "Lattice-based adaptive random testing," in *Proc. 20th IEEE/ACM Int. Conf. Automated Softw. Eng.,* 2005, pp. 333–336.
[38] H. Liu and T.Y. Chen, "Randomized quasi-random testing," *IEEE Trans. Comput.*, vol. 65, no. 6, pp. 1896–1909, Jun. 2016.
[39] C. Chow, T. Y. Chen, and T. H. Tse, "The art of divide and conquer: An innovative approach to improving the efficiency of adaptive random testing," in *Proc. 13th Int. Conf. Qual. Softw.*, 2013, pp. 268–275.
[40] Q. Du, V. Faber, and M. Gunzburger, "Centroidal Voronoi tessellations: Applications and algorithms," *SIAM Rev.*, vol. 41, no. 4, pp. 637–676, 1999.
[41] Software-artifact infrastructure repository. [Online]. Available: http://sir.unl.edu/portal/index.php
[42] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Softw. Eng.*, vol. 10, no. 4, pp. 405–435, 2005.
[43] T. Y. Chen, P. L. Poon, and T. H. Tse, "A choice relation framework for supporting category-partition test case generation," *IEEE Trans. Softw. Eng.*, vol. 29, no. 7, pp. 577–593, Jul. 2003.
[44] K. Rekab, H. Thompson, and W. Wei, "A multistage sequential test allocation for software reliability estimation," *IEEE Trans. Rel.*, vol. 62, no. 2, pp. 424–433, Jun. 2013.
[45] Z. Benkamra, T. Mekki, and T. Mounir, "Bayesian sequential estimation of the reliability of a parallel-series system," *Appl. Math. Comput.*, vol. 219, no. 23, pp. 10842–10852, 2013.
[46] M. Xie and B. Yang, "A study of the effect of imperfect debugging on software development cost," *IEEE Trans. Softw. Eng.*, vol. 29, no. 5, pp. 471–473, May 2003.
[47] H. Mei, D. Hao, L. Zhang, L. Zhang, J. Zhou, and G. Rothermel, "A static approach to prioritizing junit test cases," *IEEE Trans. Softw. Eng.*, vol. 38, no. 6, pp. 1258–1275, Nov.–Dec. 2012.
[48] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Trans. Softw. Eng.*, vol. 41, no. 5, pp. 507–525, May 2015.

Hanyu Pei received the B.S. degree in automation science and M.S degree in industrial engineering from Beihang University (BUAA), Beijing, China, in 2012 and 2015, respectively, and is currently working towards the joint Ph.D. degree in control science and engineering with BUAA and City University of Hong Kong, Hong Kong.

His current research interests include software cybernetics and software testing.

Kai-Yuan Cai received the B.S., M.S., and Ph.D. degrees in control science and engineering from Beihang University (BUAA), Beijing, China, in 1984, 1987, and 1991, respectively.

He has been a Full Professor in Automation Science with the Beihang University since 1995. He is currently a Cheung Kong Scholar (Chair Professor), jointly appointed by the Ministry of Education of China and the Li Ka Shing Foundation of Hong Kong in 1999. His main research interests include software testing, software reliability, reliable flight control, and software cybernetics.

Beibei Yin received the Ph.D. degree in control science and engineering from Beihang University (BUAA), Beijing, China, in 2010.

She was a Research Scholar with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA, in 2015. She is currently a Lecturer in Automation Science with Beihang University. Her main research interests include software testing, software reliability, and software cybernetics.

Aditya P. Mathur received the B.S. degree in electrical engineering and computer science, and the M.S. and Ph.D. degrees in electrical engineering from the Birla Institute of Technology and Science, Pilani, India, in 1970, 1972, and 1977, respectively.

He is currently a Full Professor with Purdue University, West Lafeyette, IN, USA, and the Singapore University of Technology and Design, Singapore. His current research interests include software quality, software reliability, and software process.

Min Xie (M'91–SM'94–F'06) received the Ph.D. degree in quality technology from Linkoping University, Linköping, Sweden, in 1987.

He is currently a Chair Professor with City University of Hong Kong, Hong Kong. He has authored or coauthored numerous refereed journal papers, and some books on quality and reliability engineering, including *Software Reliability Modelling* (World Scientific Publisher), *Weibull Models* (John Wiley), *Computing Systems Reliability* (Kluwer Academic), and *Advanced QFD Applications* (ASQ Quality Press). His current research interests include Reliability Engineering, Quality Management, Software Reliability and Applied Statistics.

Dr. Xie was the recipient of prestigious LKY Research Fellowship in 1991.