Science of Computer Programming 78 (2012) 93–116

Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

An improved method for test case prioritization by incorporating historical test case data

Alireza Khalilian, Mohammad Abdollahi Azgomi *, Yalda Fazlalizadeh

School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran

ARTICLE INFO

Article history:
Received 18 December 2010
Received in revised form 30 January 2012
Accepted 30 January 2012
Available online 8 February 2012

Keywords:
Software regression test
Test case prioritization
History-based prioritization
Historical fault detection

ABSTRACT

Test case prioritization reorders test cases from a previous version of a software system for the current release to optimize regression testing. We have previously introduced a technique for test case prioritization using historical test case performance data. The technique was based on a test case prioritization equation, which directly computes the priority of each test case using the historical information of the test case using an equation with constant coefficients. This technique was compared just with random ordering approach. In this paper, we present an enhancement of the aforementioned technique in two ways. First, we propose a new prioritization equation with variable coefficients gained according to the available historical performance data, which acts as a feedback from the previous test sessions. Second, a family of comprehensive empirical studies has been conducted to evaluate the performance of the technique. We have compared the proposed technique with our previous technique and the technique proposed by Kim and Porter. The experimental results demonstrate the effectiveness of the proposed technique in accelerating the rate of fault detection in history-based test case prioritization.

1. Introduction

Changing the software to correct faults or add new functionality can cause the existing functionality to regress, introducing new faults. To avoid such defects, one can retest software after modification, a task commonly known as *regression testing* [1]. Regression testing typically involves the re-running of all test cases [2,3]. Due to time and resource constraints, it is often costly and sometimes even infeasible to continually execute all the test cases of a suite on each iteration of regression testing [4,5]. Hence, regression testing is considered as one of the most expensive parts of software maintenance [1]. It is then necessary to prioritize test cases in test suites to address the cost of regression testing [3]. Prioritization techniques schedule test cases from a previous version of the software such that test cases with a higher chance of fault detection are executed earlier [4]. The prioritization is based on certain criteria and then test cases are executed in the specified order according to time and resource limitations.

Several techniques have been proposed in [1,4,6–10], for prioritizing the existing test cases to accelerate the rate of fault detection in regression testing. A number of existing approaches rely on requirement coverage. These approaches consider prioritization as an unordered, independent and one-time model. They do not take into account the performance of test cases in the previous regression test sessions, such as the number of times a test case revealed faults.

Another related topic is that of *regression test selection* [12]. A regression test selection technique selects a subset of an existing test suite, which is necessary to test and verify the current modified version of the software. There are some factors,

* Correspondence to: School of Computer Engineering, Iran University of Science and Technology, Hengam St., Resalat Sq., Tehran 16846-13114, Iran. Fax: +98 21 77240469, +98 2173218325.

E-mail addresses: khalilian@comp.iust.ac.ir (A. Khalilian), azgomi@gmail.com, azgomi@iust.ac.ir (M. Abdollahi Azgomi), ya_alizadeh@comp.iust.ac.ir (Y. Fazlalizadeh).

0167-6423/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2012.01.006

which are effective in regression test selection [7] among them are the execution of test cases at fixed intervals or the rule-based execution of test cases. To investigate this factor, history-based regression testing was proposed by Kim and Porter [7]. This study focused on using historical data for prioritizing test cases and its effect on long-term performance of regression testing, performed in environments with resource and time constraints. It assigns to each test case a selection probability taken from its performance in prior regression test sessions. Then, the test cases with higher selection probabilities are executed until the testing time is exhausted. A test case that has led to fault detection in a larger number of test sessions should have a higher selection probability than those that have led to fault detection in a fewer number of test sessions. This measure is called *fault detection effectiveness*. Similarly, the *execution history* and the *coverage of program entities* are other factors determining the selection probability of a test case. The major problem with this approach is that it considers only one of the aforementioned factors, especially in a Boolean manner (0 and 1).

Recently, we have proposed a technique for test case prioritization [15]. This technique is an extension of history-based prioritizations proposed in [7], which incorporates the notion of memoryful regression testing. The proposed technique directly computes the priority of each test case using the historical information of the test case, such as the number of executions, the number of times it exposed a fault and other relevant data. The priority is calculated using an equation, which consists of three terms, each with a constant coefficient. Our initial experiments compared the proposed algorithm to a random ordering approach using the *Siemens* suite [11] and the *Space* program [11]. The experiments provided encouraging results and served as a proof of concept for the proposed approach [15].

In this paper, we propose a new approach based on our previous approach [15]. Unlike the previous technique, the new technique modifies the prioritization equation and computes the coefficients of the equation from the historical performance data. Actually, this acts as a feedback from the previous test sessions. It is reported in the literature (e.g. [11]), that feedback causes prioritization to be more effective.

The other contribution of this paper is to conduct a comprehensive empirical study. We have implemented the proposed approach and two other approaches [7,15] and performed several detailed controlled experiments on the *Siemens* suite and a case study on the *Space* program. The experiments were aimed at evaluation the performance and effectiveness of the new prioritization technique over previous techniques in early detection of faults during regression testing. The goal of the experiments was to understand the effectiveness of several different underlying parameters of the proposed technique. The obtained results showed improvements over prior approaches and provided interesting insights into the use of historical test case performance data to compute the selection probability and determine the relative importance of test cases in examining the modified software for early detection of faults.

The rest of this paper is organized as follows. In Section 2, an overview of the background is given. The related works is surveyed in Section 3. The new approach and its motivations are presented in Section 4. The detailed experimental studies with three approaches based on history-based prioritization along with the obtained results and their analysis are given in Section 5. Finally, some concluding remarks are mentioned in Section 6.

2. Background

The test case prioritization problem [16] was first introduced by Wong et al. in 1997 as a flexible method of software regression testing. It is defined as finding a permutation of test cases to maximize an award function, which reflects the prioritization goal. The formal definition of this problem can be expressed as follows [4].

2.1. Problem statement

Given a test suite, T, the set of permutations of T, PT, and f, a function from PT to the set of real numbers, find $T' \in PT$ such that $(\forall T'' \in PT)(T'' \neq T')[(f(T') \geq f(T'')]$ [4].

In the above definition, PT represents the set of all possible prioritizations (orderings) of T, and f is a function that applies to any such ordering, yields an award value for that ordering. The problem of finding optimal execution ordering for test cases is known to be NP-hard and without any deterministic solution and has exponential complexity in the worst case [6]. Thus, prioritization techniques are necessarily heuristic and produce sub-optimal results.

The specific aim of our previous work [15] and the work of Kim and Porter [7] was to find an ordering of test cases of the current suite to increase the likelihood of early fault detection in the testing process.

2.2. Definition

Kim and Porter considered the above problem as a probabilistic approach and defined the history-based test case prioritization as follows [7]:

Definition 1. Given the test suite T, and T' be a subset of all test cases ($T' \subseteq T$), then $P_{tc,t}(H_{tc}, \alpha)$ is the selection probability of each test case, tc, in the time t, and H_{tc} is a set of t time-ordered observations $\{h_1, h_2, \ldots, h_t\}$ drawn from the previous runs of tc, which shows each test case's execution history until now. This definition uses the notion of *memoryful regression testing* and *history-based test case prioritization* [7,15,10] to identify an appropriate ordering of test cases for early detection of faults during regression testing.

Definition 2. Given the test suite T, it is considered as a tuple of test cases, since the order of test cases matters in the history-based technique.

Definition 3. The selection probability of each test case in the test suite T' based on the execution history is the likelihood that it will be used in the next test session. According to this definition, it is formulated in Eq. (1) as follows:

$$\begin{aligned} &P_0 = h_1 \\ &P_k = \alpha h_k + (1-\alpha)P_{k-1} \quad 0 \le \alpha < 1 \quad k \ge 1 \end{aligned} \tag{1}$$

In the above equation, α is a smoothing constant for weighting individual history observations. It has been set as close as possible to 0 in the experiments by Kim and Porter in order to emphasize older observations [7]. History-based prioritization can be performed in different ways, based on different test histories (definitions of H_{tc}) in Eq. (1).

In order to quantify the achievement of increasing the rate of fault detection in the experiments, the APFD [4] rate is used:

Definition 4. The *average percentage of fault detection* (APFD) rate, measures the weighted average percentage of detected faults during the execution of a test suite. This rate, ranges from 0 to 100, where higher numbers indicate earlier detection of faults.

3. Related work

Several techniques, such as [1,4,6–9,11,17,18], have been proposed for test case prioritization. The prioritization techniques presented by Rothermel and Elbaum [4,11] are basically based on *testing requirement coverage* of test cases. The requirement coverage may be *total* or *additional*. For example, the total branch coverage approach schedules test cases based on the number of total branches they exercise in a descending order. The additional approach schedules test cases based on the decreasing order of the number of additional branches they exercise, that have not yet been satisfied by the earlier tests in the prioritized sequence. Experiments on these studies, as reported in [4,11], are only based on the information obtained from the executing test cases on the immediately previous version of the software.

Kim and Porter [7] introduced a technique that uses historical execution data to prioritize test cases in a regression testing process. In this technique, test cases are prioritized according to the execution history (how often a test case has been executed lately), fault detection effectiveness (how many faults the test case has detected), and the coverage of program entities (which particular requirements has been exercised by the test case).

We have recently proposed a history-based prioritization approach [15] that combines three kinds of historical information about test cases together to an equation. Then, the selection probability is obtained and test cases are scheduled in the decreasing order of their selection probabilities. The experiments on *Siemens* subject programs for our proposed approach showed noticeable improvements against the random ordering approach, in terms of average percentage of fault detection.

The technique presented by Walcott et al. [5] uses a genetic algorithm to prioritize regression test suites. The ordered suites always run within the given time budget and have the highest potential for fault detection based on derived coverage information. They have conducted an experiment to compare the results of prioritizing test suites using their own approach with random test suite prioritization. They have also compared the results with initial ordering of test suite and its reverse ordering along with fault-aware prioritization. They obtained empirical evidence that when the maximum time allotted for testing is known or small, the proposed approach performs better than the random approach or the initial ordering, in terms of fault detection rate.

Another related approach in the literature is the one proposed by Qu et al. [19], which is a black-box prioritization approach similar to ours. To apply test case prioritization in situations where the source or the binary code is unavailable, the authors presented a prioritization technique for black-box environments. They have presented the algorithms based on the history of test cases and the run-time data. The algorithms categorize test cases by the faults they have exposed and then prioritize test cases according to the results from the related test cases. The authors then have conducted experiments to evaluate their approach. The results showed improvements of the fault detection rate for regression testing in black-box environments.

Park et al. introduced a cost-cognizant test case prioritization approach [10]. This approach uses test case costs in previous tests and fault severities detected in prior test iterations as a criterion to prioritize test cases for the current release of the software. Their approach is based on the assumption that test case costs and fault severities revealed in the previous test sessions do not significantly change from one release to another.

Mirarab and Tahvildari [1] introduced an approach that prioritizes test cases based on Bayesian networks. This approach attempts to predict the probability that each test case will reveal faults and then uses these probabilities to prioritize the test suite. In order to predict this probability, they model regression testing by means of Bayesian network in order to model uncertainty in systems.

The technique introduced by Srivastava and Thiagarajan [18] prioritizes test cases in decreasing order of the number of impacted blocks that are likely to exercise by test cases. A heuristic approach determines which blocks are influenced by each test case. The other techniques presented in the literature [4,11,17] perform test case prioritization in terms of total

or additional requirements coverage information of test cases, as collected during prior iterations of software testing. In these studies, prioritized test suites outperformed their unprioritized counterparts according to the improvements in fault detection rate. However, the relative performance of these techniques was dependent on the characteristics of test suites, programs and faults.

Jeffrey and Gupta combined the ideas of relevant slices [20,21] and coverage-based prioritization techniques [4,11,17] and proposed a new approach [22] for test case prioritization that accounts for output-influencing and potentially output-influencing coverage of test cases. In this study, a heuristic determines the priority of each test case. They conducted an experiment on *Siemens* suite along with a detailed analysis of the results. Later, they extended their method [23] and proposed two new approaches. Moreover, a family of experiments has been conducted on *Siemens* suite using the three approaches to demonstrate the advantages of the two new approaches over the first one. This study also contains a detailed analysis of the program behavior on different modifications.

The topic of *test suite minimization* [24,25] is closely related to test case prioritization. Both approaches attempt to identify *important* test cases for testers to reduce the cost of software testing. Unlike prioritization techniques, these techniques permanently discard a subset of test cases while test case prioritization preserves all test cases and just reorders them in the test list.

Another related topic is *safe regression test selection* [12,13]. This technique determines a subset of test cases that are likely to change their output due to the modifications in software and so, the faults that could be revealed by the test suite are not omitted. Regression test selection differs from test case prioritization such that the latter aims to *prioritize* the execution of test cases, so, the test cases whose output are more likely to change, execute early on during the regression testing. There are some factors which are effective in regression test selection [7]. The first factor is the test selection technique, which has been investigated in previous studies [12,13]. The second factor is the application policy, which indicates the execution of test cases at fixed intervals (daily, weekly, or monthly) or rule-based execution of test cases (after each change, after final release, etc). The effects of this factor on regression test selection costs have been studied in [14]. The third factor is the effect of resource constraints and deadlines. When the testing environment is resource constraint, the developers may limit and optimize their testing effort effectively.

Yoo and Harman [36] proposed a multi-objective formulation of regression test case selection and prioritization. They have presented two versions of their formulation: A two objective version that combines coverage and cost and a three objective version that combines coverage, cost, and fault history. The authors proposed three algorithms for solving the two versions of their approach. The fault detection history used by the authors has the following characteristics: (1) It is assigned to each *test case subset* and (2) It corresponds to how many of the known, past faults this subset has revealed in the previous version. They have also conducted experiments in which they showed that their search-based approaches could outperform the greedy approach.

4. An improved history-based prioritization technique

In our previous work [15], we extended Kim and Porter's approach to prioritize regression test suites. We proposed an equation that computes the priority of each test case using historical test case performance data. As far as we know, this is the first attempt to enhance the prioritization equation by new concepts and modify the notion of execution history and fault detection effectiveness, which were originally introduced by Kim and Porter [7]. In addition, the equation proposed in our previous work [15], considers all gathered historical information of test cases during regression testing to determine the priority of each test case. Hence, it has more potential for identifying the relative importance of test cases during regression testing, which is evidenced by the experimental results in which it showed better results. These results compared the proposed approach against random ordering approach. The proposed equation in [15] has constant coefficients that adjust the value of the equation's terms. To resolve this issue, we propose a new equation that adjusts the value of its terms by variable coefficients obtained from test case historical information.

In the rest of this section, we first present an intuitive example to motivate the new approach. Then, we present the enhanced equation with variable coefficients for history-based test case prioritization. We will also present the results of extensive experiments in the next section. They have been conducted on prioritizing suites for the *Siemens* suite subjects and the *Space* program using three approaches: Kim and Porter's approach, our previous approach and the new proposed approach. In our studies, the results of the prioritized suites using our approaches outperformed their prioritized counterparts using Kim and Porter's approach in terms of improving the rate of fault detection. In addition, the results of the prioritized suites showed considerable improvements using the new equation over our previous equation in terms of early detection of faults.

4.1. Motivations

Intuitively, if a test case has detected faults in most of the previous test sessions, it is likely to reveal some faults in subsequent testing sessions. This is because the execution trace of this test case often is likely to exercise the modifications made in the subsequent releases of the software. Therefore, the number of faults detected by a test case is effective in determining its priority in the next test sessions. Consider two test cases, tc_A and tc_B in a test suite, such that tc_A leads

```
B1:   if (a>0)                    B3:   if (c>0)
2:          x=2;                  B4:     if (d>0)
3:    else                        B5:       if (e>0)
4:       begin                    12:         output (1/(b+1));
5:       x=5;                     13:       else
B2:     if (b>0)                   14:         output (1/(y-4));
6:         y=x+1;                 15:       endif
7:       else                     16:     else
8:         y=x-1;                 17:       output (1/(x-5));
9:       endif                    18:     endif
10:    end                        19:   else
11:   endif                       20:     output (1/(x-5));
                                  21:   endif
```

Fig. 1. The example program.

to nine program failures in 20 test executions and tc_B leads to eight program failures in 16 test executions. Although, the number of faults detected by tc_A is more than tc_B, but tc_B is better to have higher priority, because it could detect faults in 50% of its executions while this is less than 50% for tc_A. Therefore, it seems that, for a test case, the ratio of the number of faults detected to the number of executions is an appropriate factor in determining the priority of that test case. A test case with higher value of this ratio can have a higher priority as compared to a test case with less value.

As mentioned earlier, time and resource constraints may prevent some test cases from execution on each test session. If some test cases never execute due to these constraints, the likelihood of missing some faults will increase until the testing process loses its effectiveness. This is mainly because each test case traverses a particular execution profile of the program. Hence, it is necessary to cycle through all test cases over multiple test sessions to assure the attainment of a certain degree of quality of the software. To achieve this, it is possible to assign some *counters* to each test case. These counters hold the number of times a test case does not execute and the number of times it could detect some faults. Analysis of this *cumulative* information, demonstrates the overall effectiveness of a test case from the beginning of the testing process. We use this information inside the terms of an equation to obtain the priority of a test case in the subsequent test session. In this way, the behavior of all test cases can be compared together. This consequently leads to an optimized and effective reordering of test cases for the next test session. Increasing the value of the counters would lead to an increase or decrease in the test cases priority.

To illustrate the above intuitions, consider the example program in Fig. 1 and suppose a branch-coverage adequate test suite given in Table 1. The sample program has four division-by-zero errors at lines 12, 14, 17, and 20. Suppose that the time constraints allow testers to run about 40% of test cases in each test session, which amounts to three test cases in this example. As shown in Table 2 for three test sessions, the executed test cases are in bold face and the underlined test cases indicate those that could reveal some faults. This table shows the prioritization using the above mentioned approach (the lower section) and that of Kim and Porter's approach (the upper section). For comparison, we have also shown the result of prioritization using our previous approach [15] (the middle section).

We first examine the result of prioritizing test cases using the Kim and Porter's approach. To apply this approach, we have used Eq. (1). In this equation, the coefficient α has been set to 0.5 to give equal weight to the test case history and its performance in the most recent test sessions. This approach requires a single definition for the parameter h_k. We observed that for each executed test case in each test session, this parameter takes the value 0; otherwise, it takes the value 1. The upper part of Table 2 shows the results of prioritization. In the first session, all the test cases have the same selection probability equal to 1, because none of them have been executed so far. In this case, the first three test cases 1, 2 and 3 would be executed. Among these tests, test case 2 exposes a fault in line 17. For the executed test cases, h_k takes the value 0 yielding the selection probability of 0.5 in the next test sessions. For other test cases, the selection probability remains unchanged, because they have not been executed. The procedure is repeated in the second test session by executing the test cases 4, 5 and 6, which now have the higher selection probabilities as compared to the others. In this session, two test cases 4 and 6 detect faults. Finally, in the third session, only the test case 7 leads to detect faults.

In order to compare the results, we have used the equation of our previous approach [15] to prioritize the test cases for the sample program. The mentioned equation needs three coefficients, denoted by α, β, and γ, which balance values of the terms execution history, the last priority of the test case, and historical fault detection effectiveness correspondingly in the equation in question. We have set all three coefficients to 0.5 to give equal weights to the different terms in the equation and make it similar to the Kim and Porter's equation we have already illustrated. In addition, we have used the percentage of branch coverage for prioritizing at first session. The results of prioritization are depicted in the middle section of the Table 2. As can be shown from this table, all of the four faults are exposed in the three test sessions, which is similar to the Kim and Porter's approach. The first round of testing now is different than Kim and Porter due to the usage of the percent of branch coverage of each test case at first test session. However, the *rate* of fault detection has been increased. It means that more test cases are exposed at earlier test sessions. Two test cases 4 and 7 lead to fault detection in the first test session, and the other two faults are exposed by the test cases 2 and 6 in the second and third test sessions respectively.

Table 1
A branch coverage adequate test suite for the program of Fig. 1.

Test case	Input (a, b, c, d, e)	B^T_1	B^F_1	B^T_2	B^F_2	B^T_3	B^F_3	B^T_4	B^F_4	B^T_5	B^F_5	Branch coverage	Fault may be detected
1	(1, 1, −1, 0, 0)	X					X					20%	–
2	(−1, −1, 1, −1, 0)		X		X	X			X			40%	17
3	(−1, 1, 1, 1, 1)		X	X		X		X		X		50%	–
4	(−1, −1, 1, 1, 1)		X		X	X		X		X		50%	12
5	(1, −1, −1, −1, −1)	X					X					20%	–
6	(−1, 1, −1, 1, 0)		X	X			X					30%	20
7	(0, 0, 1, 1, −1)		X		X	X		X			X	50%	14

Table 2
Test case priorities during four test sessions using Kim and Porter's approach (upper section), our previous approach (middle section), and our proposed approach (lower section).

Approach	Regression test session	Test case priority
Kim and Porter's approach	1	[**1**, **2**, **3**, 4, 5, 6, 7]
	2	[**4**, **5**, **6**, 7, 1, 2, 3]
	3	[**7**, **1**, **2**, 3, 4, 5, 6]
Our previous approach	1	[**3**, **4**, **7**, 2, 6, 1, 5]
	2	[**4**, **7**, **2**, 6, 1, 5, 3]
	3	[**6**, **1**, **5**, 2, 3, 4, 7]
The proposed approach	1	[**3**, **4**, **7**, 2, 6, 1, 5]
	2	[**2**, **6**, **1**, 5, 4, 7, 3]
	3	[**5**, **4**, **7**, 3, 2, 6, 1]

Now, we use the above example to motivate the approach proposed in the rest of this section, which attempts to use all historical data together to prioritize test cases effectively. The problem with our previous equation [15] is that it uses constant coefficients to control the effect of each term against the other. In this case, each term always has a same effect on prioritizing a certain test case. In a test session, the execution history of a test case should be dominant determiner in prioritization, and for the other test session, the fault detection effectiveness may have the major impact. Suppose that test cases are ordered according to the percentage of branch coverage in the first test session. This is a criterion to discriminate the test cases when there is no historical data. According to this criterion in the first session, the test cases 3, 4, and 7 are executed and the test cases 4 and 7 reveal faults in the lines 12 and 14. Since the h_k takes the value 0 for the executed test cases, their selection probability will decrease. Moreover, the test case 3 gets a lower selection probability than the two others, because it was executed without detecting any faults. The increment of h_k to 1 for unexecuted test cases increases their selection probability. Note that this parameter is a *counter* and differs from that of Kim and Porter's approach where h_k was a Boolean value.

In the second test session two test cases (2 and 6) out of three, detected faults at lines 17 and 20. The procedure of increment and decrement in the selection probabilities follows a similar way as in the first session. Again the test case 5 is not executed and the value of h_k for this test case increases to 2. Hence, its selection probability increases to be greater than all test cases. By the end of the third test session, all of test cases have been executed at least once.

The overall results indicate that our proposed approach shows a higher rate of fault detection in comparison with Kim and Porter's approach and our previous approach, according to the APFD metric. In addition, the faults were detected earlier during the test sessions. This example illustrates the benefits of using different factors together to increase the effectiveness of test case prioritization. The effectiveness is achieved through the following:

1. Increasing the fault detection efficacy in each test session,
2. Early detection of faults over continuous test sessions, and
3. Early cycling through all test cases during test sessions when some of the test cases are not always executed in each test session.

4.2. The proposed approach

The example, presented in the previous subsection, motivated us to propose a new approach. It is worth mentioning that we consider regression testing as follows:

1. An ordered sequence of testing sessions, such that each session may be dependent on the previous testing session, and
2. The time and resource constraints, which prevents testers from performing an exhaustive testing (rerunning all existing test suites).

Now, we propose the following heuristic for history-based test case prioritization. In this approach, the priority of each test case is directly computed by means of an equation. The equation uses the historical test case data and is composed of the following three parts:

1. The first part is the execution history. This factor is used to ensure that regression testing process cycle through all test cases and prevents from long time discarding a test case from running over consecutive regression test sessions. Kim and Porter have considered this factor as a Boolean value [7]. In our equation, it acts as a positive integer counter and increases by one each time a test case does not execute. Once the test case is executed, it is reset to zero. Such definition of execution history puts unexecuted test cases forward with an increasing rate in the next test session. In the context of operating systems and inter-process communication, sleep and wakeup primitives were used for process scheduling. However, there were many issues until Dijkstra thought to count wakeup signals, the concept of semaphores, which was an important and novel idea. The execution history here has a similar philosophy.
2. The second part is the test case priority in the previous test session. This factor is used to smooth the changes in the priority values from one test session to another.
3. The third part is historical fault detection effectiveness. This is the ratio of the number of times the test case detects a fault to the total number of its executions during regression test iterations. This would indicate the relative effectiveness of a test case. The priority of a test case in each session is proportional to this ratio.

One of the differences between prioritization techniques is their optimization algorithm. Optimization means that selection of test cases in the ordered list would be done such that the score function is optimized. If the APFD rate is considered as the score function, then optimization means to obtain values as close to 100 as possible. When ordering test cases according to some criteria (coverage information, historical selection probability, Bayesian network selection probability, *etc.*) a feedback mechanism could be used. For coverage-based techniques, feedback means that when adding a test case to the ordered list, the effect of test cases already added to the list are taken into account. Because this prevents the technique from selecting a test case that has a similar behavior to the already added test cases in the ordered list according to some criteria. For our history-based approach, a new feedback mechanism is proposed. When determining a test case selection probability, we take into account its execution history and fault history within all previous regression test sessions.

The following equation computes a test case's priority by adding the above three factors. Similar to the Kim and Porter equation [7], a coefficient is used with each factor to control the effect of each factor and also enable us to add these factors together. After prioritization, test cases are executed according to the available testing time and resources. The prioritization equation for each test case, which is presented in our previous work [15], is as follows[1]:

$$\boxed{\begin{array}{l} PR_0 = \textit{The percentage of code coverage of the test case} \\ PR_k = \alpha h_k + \beta PR_{k-1} + \gamma HFDE_k. \quad 0 \le \alpha, \beta, \gamma < 1 \quad k \ge 1 \end{array}} \tag{2}$$

$$HFDE_k = \begin{cases} 0 & \textit{if tc has not been executed yet} \\ \frac{fc_k}{ec_k} & \textit{otherwise} \end{cases} \tag{3}$$

$$fc_k = \sum_{i=1}^{k-1} f_i \,.\, f_i = \begin{cases} 1 & \textit{if tc has revealed fault(s) in test session } i \\ 0 & \textit{otherwise} \end{cases} \tag{4}$$

$$ec_k = \sum_{i=1}^{k-1} e_i \,.\, e_i = \begin{cases} 1 & \textit{if tc has been executed in test session } i \\ 0 & \textit{otherwise} \end{cases} \tag{5}$$

$$h_k = \begin{cases} 0 & \textit{if tc has been executed in test session } k-1 \\ h_{k-1} + 1 & \textit{otherwise}. \end{cases} \tag{6}$$

In Eq. (2), PR_k is the priority of test case in the next test session, $HFDE_k$ is the test case historical fault detection effectiveness, h_k is the test case's execution history, and PR_{k-1} is test case priority in the current test session. Furthermore, this equation states that PR_0 is the percentage of the code coverage of the test case. With this approach, we have three choices for PR_0:

1. Using the percent of code coverage with respect to any coverage criterion.
2. Using any other approach to prioritize test cases at first test session.
3. Executing test cases randomly until the testing time is exhausted.

The existence of a code coverage report on a per test case basis or using any other prioritization approach at the first test session would enhance this black-box approach to establish the relative ordering of test cases more precisely. There are available tools to instrument the source code of programs automatically in order to gather the coverage data [12,26–28]. Hence, using code coverage is reasonable and practical in real situations. In the case of adding a new test case in

[1] We call this technique the constant coefficient (CC) prioritization technique.

test suite, this approach records the coverage data for the new test case and obtains its current priority through the recursive equation (2).

A problem concerning Eq. (2) is its coefficients that are constant values. This means that the effect of the different terms of the equation will be the same in all regression test sessions. But according to the historical data and circumstances in each test session, each term must have its proper effect on determining the priority of each test case, neither negligible, nor dominant, which is evidenced by the empirical studies in [15]. Since the value of PR_k can be greater than one, it will gradually increase during consecutive test sessions. This may consequently lead to it dominating other factors. Thus, the coefficient of β should be decreased to balance the effect of PR_{k-1} against the other terms of the equation. Moreover, by selecting a small value for α (close to zero), the effect of h_k for low values would be negligible. In contrast, selecting a greater value for α (closer to the upper bound, 1) can dominate the effect of the other terms for greater values of h_k. Therefore, the effect of each factor must be controlled against the conditions. To achieve this, the coefficient of each factor should be accurately determined in each test session and then should be used in the prioritization equation.

One way to determine the coefficients, is applying the available information from all previous test sessions. This information can be extracted from the test case executions and their fault detection effectiveness. For example, consider two test cases *A* and *B* with the equal priorities in the current test session. Suppose that both test cases are executed, but only the test case *A* exposes fault. In the next test session, the priorities of both test cases should be decreased, but that of test case *A* should be decreased less than that of test case *B*. This can be achieved using a lower coefficient for the test case *B* less than for test case *A*. Guided by this intuition and by building on the history-based approach, Eq. (2) is modified such that the coefficients of the prioritization equation are determined through computation by using the historical information of execution and fault detection of each test case. We call this the *Variable Coefficient* (VC) technique for prioritization.

Based on the above discussion, the new prioritization equation for each test case in the *k*-th test session can be rewritten as follows:

$$\begin{aligned} PR_0 &= \textit{The percentage of code coverage of the test case with respect to } n \\ &\quad \textit{different coverage criteria} \mid n \\ PR_k &= (\alpha h_k + \beta PR_{k-1})/k. \quad 0 \le \alpha,\ \beta < 1 \quad k \ge 1 \end{aligned} \tag{7}$$

$$\alpha = \left(1 - \left(\frac{fc_k + 1}{ec_k + 1}\right)^2\right)^{h_k} \tag{8}$$

$$\beta = \left(\frac{fc_k + 1}{ec_k + 1}\right)^x \qquad x = \begin{cases} 1 & \textit{the test case has revealed some faults} \\ 2 & \textit{the test case has not revealed any fault.} \end{cases} \tag{9}$$

In Eq. (7), h_k is the same as in Eq. (2) and is defined in Eq. (6). Moreover, ec_k and fc_k are defined in the Eqs. (4) and (5), respectively. In Eq. (7), PR_0 is defined for each test case as the percentage of code coverage of the test case with respect to *n* various coverage criteria divided by *n*. Thus, the influence of test case code coverage may propagate to the next prioritizations due to the recurrence in the equation. Note that this definition of P_0 is orthogonal to the main contribution of this paper and it can also be done in the Eq. (2) with no conceptual problem.

The following modifications and improvements are made to Eq. (2) to obtain Eq. (7):

1. In Eq. (7), the term $HFDE_k$ has been removed as compared to Eq. (2). The coefficients α and β now take into account the information provided by $HFDE_k$.
2. Eq. (7) is now more similar to Eq. (1) that was presented by Kim and Porter [7]. We have execution history (it is a positive integer counter) and the test case selection probability in the previous session.
3. The value of PR_k in Eq. (2), determines the priority of each test case, but it may be greater than one. Unlike this, the PR_k value in Eq. (7) is divided by *k*, the number of current test sessions. This will normalize its value and make it the test case selection probability as in Eq. (1).
4. The Eq. (7) is intended to prioritize each test case by considering all available historical factors in different situations that, for a test case, may occur during consecutive test sessions. One way to achieve this goal is to set an appropriate coefficient for each term of Eq. (7). There are different choices:
   a. A constant coefficient for each term independent of the other coefficient.
   b. Defining constant coefficient for each term, that one is the inverse of the other (for example α and (1 - α)).
   c. Defining variant coefficient for each term such that it reflects the functionality of that test case in terms of its fault detection effectiveness. Variable coefficients can control the effect of one term against the other term according to historical performance of each test case. This definition of coefficients is applied for our proposed equation.
5. To develop the required coefficients, we were guided by some intuitions for finding the equilibrium and learning in game theory [37].
6. The aim of the proposed equation has been to produce a smaller selection probability value for a test case that is, in some way, less important than the other test cases. The generated selection probabilities of test cases should be semantically comparable to each other. To this end, they depend on multiple kinds of historical information.

It is possible to use various control-flow or data-flow coverage criteria, such as statement, branch or definition-use pair to measure the test case adequacy [3,11,29]. In addition, there are other test coverage criteria in the literature [29], which are also acceptable. However, an appropriate test criterion should be measured cost-effectively in practice. Using more than one coverage criteria can break ties in percentage of code coverage which may occur among test cases. Moreover, in code-based prioritization techniques, researchers empirically showed [3,4,30] that the code coverage of each test case is an appropriate approximation of its fault detection capability. Although high coverage does not always imply high fault detection effectiveness, but useful information included in coverage criteria helps history-based prioritization as a measure in the first session to distinguish test cases where there is no test case history.

If the coverage information used in the first session is outdated, the test cases may be incorrectly discriminated. This consequently may lead to an inefficient prioritization in the first test session. But the efficiency of test cases in fault detection changes the variables of the prioritization equation in next sessions. The result is to gradually improve the effectiveness of prioritization even in absence of the updated coverage information. This does not mean that coverage information needs rarely or never be updated.

When a test case is executed, the value of h_k changes to zero for that test case. In this case, the value of β and PR_{k-1} will determine the priority of test case in the next test session. The value of β will be computed according to Eq. (9). Since the value within the parenthesis is less than or equal to 1, the selection probability and priority of the executed test case would be decreased. Note that the value inside the parenthesis is higher for test cases that are more effective with respect to fault detection. Based on whether it could reveal some faults in the previous session, the value within the parenthesis will be raised to the power of 1 or 2. Squaring a value less than one gives a result that is smaller. Therefore, the amount of decrease in selection probability for an ineffective test case with respect to fault detection is greater than an effective one.

If a test case is not executed in a test session while it was executed in the previous session, the value of h_k will be increased by 1 beginning from 0. This continues in the next sessions until it is finally executed in a session where it will change to 0. Hence, the value of h_k is a positive integer. In order to control the effect of h_k against PR_{k-1}, the value within the outer parenthesis in Eq. (8) is raised to the power of h_k to make it smaller with respect to increase in the value of h_k.

The procedure of prioritization with the new proposed equation is similar to that used with our previous work [15] and is as follows:

1. The test case coverage data is gathered with respect to n coverage criteria.
2. The first prioritization is carried out by sorting the test cases in terms of their percentage of code coverage in an ascending order.
3. The test cases with higher percentage are executed.
4. In the subsequent test sessions:
   a. The selection probability of each test case is computed according to Eq. (7).
   b. Based on available testing time and resources, test cases with higher values of selection probabilities are selected to run.
   c. The historical information is recorded.

5. Experimental studies

In order to evaluate the proposed approach, we have conducted several empirical studies. These studies include controlled experiments with *Siemens* suite and a case study using the *Space* program. In this section, we explain these studies.

5.1. Experimental setup

In our experiments, we have used eight C programs as subjects (Table 3), which are available in [31]. Seven of these programs, used in our controlled experiments, belong to *Siemens* suite. These programs were assembled by researchers at Siemens Corporate Research for experiments with control-flow and data-flow test adequacy criteria [11]. The eighth one, used for our case study, is the Space program. The Space program, which was developed for the European Space Agency, is an interpreter for an array definition language (ADL). Each program is associated with a set of faulty versions (each contains a seeded error) and the pool of test cases.

The goal of the experiments is to investigate how well the proposed technique prioritizes test cases in terms of the rate of fault detection as compared to the previous history-based techniques. The experimental approach is to obtain various test suites, prioritize their test cases according to various history-based prioritization techniques and then measure the resulting APFD value for each prioritized test suite. Basically, the APFD metric [3] is defined as follows:

$$APFD = 1 - \frac{TF_1 + TF_2 + \cdots + TF_m}{nm} + \frac{1}{2n} \tag{10}$$

where, n is the number of test cases and m is the number of existing faults in the software. Each TF_i in this equation shows the place of a test case in the ordered suite that first reveals the fault i. This metric measures the weighted average percentage of detected faults over the life of the test suite. The values of this metric are between 0 and 100 and the higher values indicate faster (better) fault detection.

Table 3
Subject programs used in the experiments.

Program name	LOC	#Faulty versions	Test case pool size	Program description
ptok	402	7	4130	Lexical analyzer
ptok2	483	10	4115	Lexical analyzer
replace	516	32	5542	Pattern substitute
sched	299	9	2650	Priority scheduler
sched2	297	10	2710	Priority scheduler
tcas	148	41	1608	Altitude Separation
totinfo	346	23	1052	Info accumulator
space	6218	38	13585	ADL interpreter

Our experiments follow a setup similar to that used by Elbaum et al. [11] and also used in our previous study [15]. We used three groups of test suites [31] that were generated for each program. These groups are branch coverage adequate, definition-use[2] coverage adequate and randomly selected test suites. These suites were generated from the provided test pools for each program. The procedure for creating branch coverage adequate test suites is as follows: test cases are picked from the test pool randomly for each branch that the test case exercises that branch. The resulting suite is branch coverage adequate but not in a minimal manner. One thousand suites were generated in this way. Besides, 1000 test suites were made similarly for definition-use coverage adequacy. In addition to the above suites, 1000 randomly generated suites were prepared such that test cases are selected randomly to have the same size as their corresponding branch coverage adequate test suites.

In our experiments, we needed to obtain the coverage information of test cases for three coverage criteria. To achieve this, the total set of exercised branches, all-uses and function entries of each test case were measured using the instrumented version of each subject program. Each subject program was hand-instrumented for branch coverage. Instrumentation for all-uses and function entry coverage was measured using the ATAC tool [26].

As mentioned above, each subject program has several single fault versions. In order to identify the types of seeded errors in single fault versions, each faulty version of each program was examined. As a result, six different categories of seeded faults were identified: (1) operator, (2) operand, (3) value of a constant, (4) removed code, (5) adding code, and (6) changing the logical behavior of code (usually involving a few of other types of faults simultaneously). Thus, the faults used in the experiments, consist of a wide range of fault types. For evaluating prioritization techniques, one needs versions with varying numbers of faults for each program. To this end, a set of multi-fault versions composed of non-interfering single faults, have been created by Elbaum et al. [11]. We randomly selected 29 multi-fault versions of each program in order to simulate 29 sessions of regression testing to study the proposed approach's performance during continuous executions. The 29 versions are selected because it is the minimum number of non-interfering multi-fault versions that can be generated using single fault versions [11]. For each version, we have executed the proposed approach, the previous approach [15] and Kim and Porter's approach [7] on various 1000 test suites, each suite containing different test cases from the test pool. To save the space and have a general view of the results, we plotted an *all versions* boxplot[3] for each program. The boxplots were created using the SAS statistical package [32].

Given the sets of different coverage data for each test case along with three distinct groups of test suites and also the set of multi-fault versions for each program indicating 29 consecutive versions of software in regression testing process, we used this information to prioritize test cases according to various approaches including our new proposed approach (VC), the previously proposed approach [15] (CC), and Kim and Porter's approach [7] (hereafter, we refer to this approach as KP). Afterwards, the APFD value for each prioritized test suite was computed in order to evaluate its rate of fault detection. The fault detection information for each subject program and each faulty version was obtained from [31].

5.2. Experiment I: the proposed approach (VC) vs. the KP approach

This experiment was carried out to compare the APFD results computed from the 1000 prioritized branch coverage adequate test suites using the proposed approach to that of KP approach across 29 multi-fault versions of each subject program resulting in the 29 consecutive regression test sessions. In this experiment, only 30% of test cases within each test suite were executed.

The value of P_0 in the proposed approach is the percentage of the branch coverage of each test case. The prioritization equation of the KP approach involves a coefficient, α, to control the effect of each part of the equation. To have a

[2] Definition-use pair association is a coverage criterion derived from data-flow information that begins from a definition of a variable and ends with a use of the same variable (Zhu et al. 2007).

[3] Boxplot diagrams are commonly used to visualize the empirical results in test case prioritization studies. In these boxes, each distribution of data sets is shown using a box and a pair of whiskers. The height of the box spans the central 50% of data. The upper and lower end of the box shows its upper and lower quartile. The horizontal line within the box marks the median. The whiskers are vertical lines attached to the box and extend to the smallest and largest points of data which are within the outlier cutoff.

Table 4
Average suite sizes and APFD values for each subject program using the branch coverage adequate suites, when prioritizing according to the VC approach and KP approach.

Program name	Suite size	VC	KP				
			0.1	0.3	0.5	0.7	0.9
ptok	318.24	83.39	58.47	58.47	58.80	59.36	58.74
ptok2	389.44	93.73	71.96	71.96	72.43	74.23	76.06
replace	397.97	64.39	51.58	51.58	51.38	51.26	49.90
sched	224.31	69.03	52.23	52.23	52.17	52.58	53.17
sched2	233.64	68.09	43.32	43.32	43.07	43.96	46.49
space	4362.35	91.70	89.44	89.44	89.44	89.53	86.84
tcas	83.19	42.14	44.20	44.20	44.59	45.40	44.28
totinfo	198.05	89.06	72.22	72.22	71.88	71.58	71.52

comprehensive comparison, the KP approach has been executed for five values of α, including 0.1, 0.3, 0.5, 0.7 and 0.9. We have instantiated the KP approach with h_k defined as the execution history. For each test session i, in which a test case is executed, h_k takes the value 0; otherwise it takes 1.

Table 4 gives the results for both approaches. In addition, the table lists the average suite size among the suites generated for each subject program. Note that larger programs (such as *replace* and *space*) have also larger test suites on average, because more test cases are required to make the suite's branch coverage adequate.

As shown in Table 4, in all cases except *tcas* the VC prioritized suites consistently results in improved average APFD values as compared to their KP approach counterparts in the different ranges. For *tcas*, the results are slightly degraded on average using the VC approach as compared to KP. The reason is that the *tcas* program is the simplest of the *Siemens* suite with fewest branches among other programs. As a result, many of test cases that exercise different execution paths within the program may still map the test cases to a same output. Therefore, *tcas* has much potential that traversing a fault by a test case may lead to the same correct output. Since the percent of branch coverage has been used for P_0 in the prioritization equation, our approach has less potential to discriminate test cases precisely and this is why the average APFD values are degraded as compared to the KP approach. In the remaining cases, the amount of average percentage of improvement is high, except for space that the improvement is relatively small.

To determine whether the improvement in average APFD values when using the VC approach over the KP approach is statistically significant, we conducted a *hypothesis test for the means of two samples*[4] [33]. The 1000 APFD values obtained by each of the two approaches are considered as the samples. We consider the *null hypothesis* to be that is no difference in the mean APFD value of both the VC and KP prioritized suites. We used a reference table of critical values presented by Freund [33] to calculate the percentage of rejecting the null hypothesis.

Tables 5 and 6 show the computed *z values* for the *hypothesis test*. As can be seen, the percentage of confidence with which we may reject the null hypothesis is greater than 99.9% for all ranges and programs. Note that the larger the magnitude of the computed *z value*, the greater confidence in rejecting the null hypothesis. Thus, from the tables we observe that in all cases where the average APFD value has improved by using VC over KP, the amount of the improvement is statistically significant except for *tcas*, in which the results were degraded. This is shown by the fact that the magnitude of the *z value* for *tcas* is much smaller than other programs.

The reason that VC is able to show consistent average improvement over KP in nearly all cases is because VC takes into account the historical performance information of each test case in all previous test sessions, instead of just the data from previous test session. It is expected that the historical performance information of each test case has the potential to determine the test cases with higher likelihood of fault detection. This information includes: (1) the number of executions, (2) the number of fault detections, and (3) the number of recent test sessions in which the test case has not been executed due to the time and resource constraints. The fact that the obtained results show near-consistent average improvement when going from the VC approach to the KP approach is promising for the potential of the proposed approach, and shows that there may be something to be gained by accounting for historical test case performance data in all previous test sessions during test case prioritization.

Fig. 2 presents boxplots to gain insight into how much benefit the new approach provides for the suites individually. It illustrates the benefit of the VC approach over the KP approach in terms of promoting improved APFD values for prioritized suites. Specifically, the figure shows the results of the APFD values by the VC approach and KP approach in five ranges for the given program. In each plot, the white boxplots represent the VC approach and the gray boxplots represent the KP approach in the ranges 0.1, 0.3, 0.5, 0.7 and 0.9 from left to right, respectively. The higher the place of the boxplot, the faster the prioritization technique reveals faults. In all cases, except for *tcas*, the VC approach provides considerable improvements, as the boxplots in Fig. 2 show, both in fault detection and stability of the results for various test suites. This implies that the

[4] This is a statistical method for determining whether there may be any statistically-significant difference between the means of two populations, given samples. The procedure is to formulate a *null hypothesis* that assumes the population means are identical, then compute a *z value* from the two samples which is referenced in a table of critical values to determine the confidence with which we may reject the null hypothesis.

Table 5
Computed z values when comparing the APFD values of the suites prioritized using VC and KP for 3 different values of α, 0.1, 0.3, and 0.5.

Program name	0.1		0.3		0.5	
	z	%conf	z	%conf	z	%conf
ptok	117.23	>99.9%	117.23	>99.9%	115.46	>99.9%
ptok2	127.70	>99.9%	127.70	>99.9%	125.48	>99.9%
replace	44.73	>99.9%	44.73	>99.9%	45.48	>99.9%
sched	54.99	>99.9%	54.99	>99.9%	55.35	>99.9%
sched2	88.79	>99.9%	88.79	>99.9%	90.14	>99.9%
space	13.89	>99.9%	13.89	>99.9%	13.88	>99.9%
tcas	−7.40	>99.9%	−7.40	>99.9%	−8.77	>99.9%
totinfo	114.19	>99.9%	114.19	>99.9%	116.57	>99.9%

Table 6
Computed z values when comparing the APFD values of the suites prioritized using VC and KP for 2 different values of α, 0.7, and 0.9.

Program name	0.7		0.9	
	z	%conf	z	%conf
ptok	112.18	>99.9%	115.51	>99.9%
ptok2	117.60	>99.9%	118.98	>99.9%
replace	45.90	>99.9%	52.35	>99.9%
sched	54.18	>99.9%	53.03	>99.9%
sched2	86.67	>99.9%	79.36	>99.9%
space	13.34	>99.9%	29.46	>99.9%
tcas	−11.61	>99.9%	−7.73	>99.9%
totinfo	119.14	>99.9%	124.74	>99.9%

VC approach can produce reasonable orderings which lead to faster fault detection of test cases. It is interesting to note that the average APFD results for the *tcas* are nearly same for both approaches. The following observations can be made from the plots of each program except for *tcas*. The upper quartile of the VC boxplot is smaller than those of KP approach. Further, the lower quartile is much smaller and has a higher position. In addition, the median of the APFD values in VC plot has a considerable higher position. By these observations, we may conclude that cases the VC approach leads to an improvement over KP is significantly more than those where results are degraded. Moreover, these results are consistent with earlier average APFD and computed z values when using the VC approach.

A final note about the results is that the KP data often has a much lower bound than VC data. This relates to two facts: (1) The VC approach uses the percent of branch coverage for prioritization at first session, while the KP approach has no such data and runs test cases as much as needed from the beginning of the test suite. The percent of branch coverage of each test case is an appropriate approximation of the effectiveness in fault detection for that test case [3,4,30]. (2) The VC approach considers both the execution history and fault detection effectiveness for each test case cumulatively, while the KP approach considers only execution history as a Boolean value.

5.3. Experiment II: prioritization using VC, CC and KP approaches with varying percentages of executed test cases

When evaluating the VC approach, it may be more effective to compare the prioritization results using VC approach with those of CC and KP approach when executing different fractions of the test suite rather than executing a fixed percent of the suite. The intuition is that the performance of these history-based approaches may vary with different fractions of executed test cases. This idea is analyzed in this subsection.

Given the above intuition, 1000 branch coverage adequate test suites were prioritized using three approaches: VC, CC and KP. The goal of this experiment is to investigate how well the introduced history-based approaches act with different fractions of test case execution. To this end, the experiment was conducted on five different test suite fractions in the ranges 5%, 10%, 15%, 20% and 30%. Similar to the first experiment, prioritization was performed across 29 multi-fault versions of each subject program resulting in 29 continuous regression test sessions. In this experiment, the value of P_0 for both the VC and CC approaches was considered to be the percentage of branch coverage of each test case. Additionally, the α coefficient for the KP approach was set to the value 0.5, which has been shown in the previous experiment to provide a reasonable balance in the prioritization equation. Moreover, the coefficients of the equation for CC approach have been set to 0.1, 0.5 and 0.5, respectively (based on the experimental results presented in [15]).

Table 7 gives the results of conducting the experiment. Similar to Table 4, the average APFD values of the prioritized suites using the VC, CC and KP approach in five suite ranges are presented. Moreover, the average suite size of each subject program among all suites is listed in Table 7. As shown in this table, in all cases, except *tcas*, *sched* and *sched2*, the VC prioritized suites

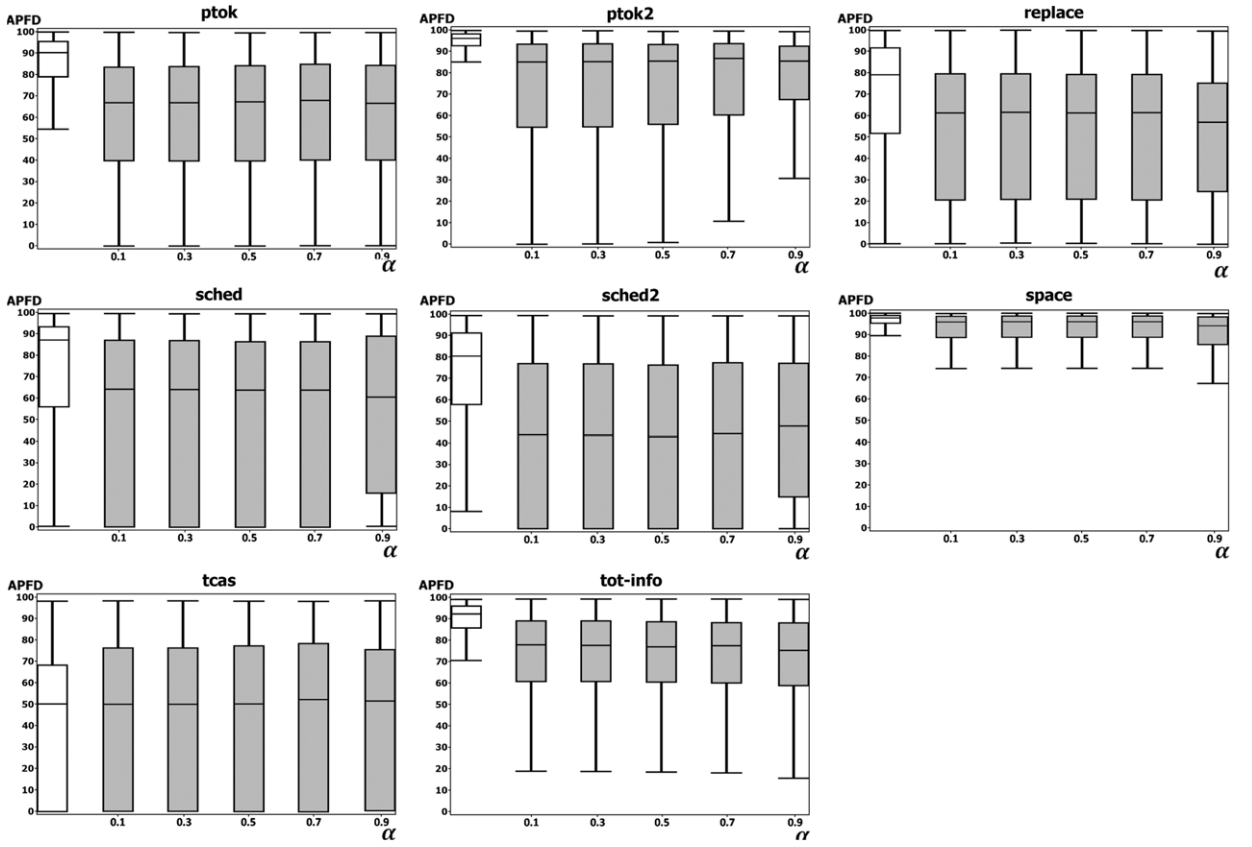

Fig. 2. The benefit of the VC approach over the KP approach in the first experiment.

consistently result in improved average APFD values as compared to their CC approach counterparts in different ranges. The VC prioritized suites for the *sched* and *sched2* programs also result in improved average APFD values as compared to their CC approach counterparts in all ranges, except for the range 20%, in which the *sched* (slightly) and *sched2* (relatively) are degraded. For *tcas*, the results are degraded on average using the VC approach as compared to CC in all ranges, except for the range 30%, in which the results are slightly degraded. In the remaining cases, the amount of average percentage of improvement is high. In addition, the table shows that in all cases except for *tcas* in the ranges 5%, 10%, 20% and 30% in which the results are slightly degraded, the VC prioritized suites consistently result in improved average APFD values as compared to their KP approach counterparts in different ranges. Further, the amount of the improvement is significantly high (between 30% and 40%), except for the *space*, where the improvement is moderate (between 7% and 10%).

Tables 8 and 9 show the results of conducting a *z test* to determine whether the improvement in the average APFD values is statistically significant for (1) when using the VC approach over the CC, and (2) when using the VC approach over KP approach. The percentage of confidence with which we may reject the null hypothesis is greater than 99.9 for all programs and for all fractions of test suites in both tables, except for *sched* in the range 20%, which was greater than 98.8.

As shown in Table 8, in all cases where the average APFD value is improved when using VC over CC, the amount of improvement is statistically significant in all cases except for the *tcas* in all ranges and the *sched* and *sched2* in the range 20%, in which the results were degraded. However, the magnitude of the *z value* for *sched* in the range 20% is much smaller than for *sched2* in the corresponding range and *tcas* in all ranges.

As shown in Table 9, in all cases where the average APFD value was improved when using VC over KP, the amount of improvement is statistically significant in all cases except the *tcas* in four ranges 5%, 10%, 20% and 30% in which the results were degraded. However, in cases where degradation occurs, the magnitude of *z value* is much smaller than cases with improvement.

The reason that VC is able to show consistent average improvement over KP in nearly all programs and ranges is because VC also takes into account the historical performance information of each test case in all previous test sessions instead of just the data from the previous test session. Also, the improvement of VC over CC in nearly all cases is because the prioritization equation for VC uses variable coefficients rather than constant coefficients as done in the CC approach. The variable coefficient is computed from the test case performance data in all prior test sessions in addition to the test case's efficiency in the immediately previous session. Hence, each part of the equation will increase or decrease according to the test case functionality in the last and previous test sessions. But in CC approach, each factor has a constant coefficient in all

Table 7
Average suite sizes and APFD values for each subject program using the branch coverage adequate suites, when prioritizing according to the VC, CC and KP approach and the execution of five different fractions of test suite.

Program name	Suite size	Approach	Range				
			5%	10%	15%	20%	30%
ptok	318.24	VC	63.74	72.57	76.34	79.86	83.39
		CC	27.57	37.26	41.42	63.68	60.46
		KP	18.98	30.38	41.55	49.42	58.80
ptok2	389.44	VC	82.57	87.90	90.44	92.15	93.73
		CC	54.14	67.49	63.87	83.27	88.02
		KP	41.02	60.30	65.07	68.48	72.43
replace	397.97	VC	36.69	46.99	54.85	60.02	64.39
		CC	28.96	41.14	42.09	57.58	57.61
		KP	26.14	36.59	42.62	50.98	51.38
sched	224.31	VC	41.23	54.17	59.85	64.26	69.03
		CC	30.47	46.32	42.94	65.01	60.83
		KP	27.80	37.63	46.81	51.78	52.17
sched2	233.64	VC	56.52	52.70	59.46	64.58	68.09
		CC	19.37	40.47	32.84	69.19	55.03
		KP	16.84	25.16	32.03	37.05	43.07
space	4362.35	VC	86.06	90.48	91.30	92.44	91.70
		CC	84.08	84.12	77.94	85.13	80.46
		KP	78.42	83.81	82.55	84.77	89.44
tcas	83.19	VC	9.96	19.80	27.16	32.99	42.14
		CC	13.74	25.51	34.03	41.36	43.01
		KP	13.97	22.04	26.34	34.59	44.59
totinfo	198.05	VC	77.37	79.75	83.44	85.86	89.06
		CC	46.20	54.61	57.35	72.34	73.65
		KP	43.57	48.49	57.01	68.00	71.88

Table 8
The computed z values when comparing the APFD values of the suites prioritized using VC and CC for five different fractions of the test suite execution.

Program name	5%		10%		15%		20%		30%	
	z	%conf	z	%conf	z	%conf	z	%conf	z	%conf
ptok	130.12	>99.9%	134.27	>99.9%	136.08	>99.9%	69.05	>99.9%	119.55	>99.9%
ptok2	123.59	>99.9%	119.17	>99.9%	139.87	>99.9%	72.22	>99.9%	66.45	>99.9%
replace	25.99	>99.9%	18.50	>99.9%	41.70	>99.9%	7.79	>99.9%	24.46	>99.9%
sched	36.19	>99.9%	25.38	>99.9%	56.21	>99.9%	−2.51	>98.8%	30.64	>99.9%
sched2	138.36	>99.9%	44.43	>99.9%	99.56	>99.9%	−17.54	>99.9%	58.83	>99.9%
space	9.85	>99.9%	30.22	>99.9%	67.19	>99.9%	34.75	>99.9%	57.08	>99.9%
tcas	−18.35	>99.9%	−22.18	>99.9%	−24.21	>99.9%	−29.53	>99.9%	−3.31	>99.9%
totinfo	121.83	>99.9%	105.73	>99.9%	113.41	>99.9%	73.57	>99.9%	123.35	>99.9%

test sessions. This causes each part of the prioritization equation to have the same effect on total priority value of the test case, regardless of its effectiveness in prior sessions, especially the last session. Notice that the outcome of the prioritization equation for each test case using the CC approach can be greater than one and it tends to grow in consecutive sessions. Also, notice that the factor h_k grows one unit each time a test case does not execute. As a result, PR_{k-1} or h_k may dominate other factors in the prioritization equation of the CC approach, while these are controlled against the conditions in the VC prioritization equation. Therefore, the VC approach is able to determine the relative importance of each test case more accurately than the CC approach.

It is expected that using the historical performance information of each test case to control the effect of different parts of the prioritization equation has the potential to schedule test cases and put those with higher likelihood of fault detection earlier in the list. The fact that the obtained results show near-consistent average improvement when going from the VC approach to the CC and KP approaches with different fractions of the test suite execution is promising for the potential of the proposed approach in the environments with the time and resource limitations. It also shows that there may be something to be gained by determining different components of the prioritization equation through computation using historical test case performance data in all previous test sessions during the test case prioritization.

Table 9
The computed z values when comparing the APFD values of the suites prioritized using VC and KP for five different fractions of the test suite execution.

Program name	5%		10%		15%		20%		30%	
	z	%conf	z	%conf	z	%conf	z	%conf	z	%conf
ptok	170.96	>99.9%	164.32	>99.9%	137.08	>99.9%	126.04	>99.9%	115.46	>99.9%
ptok2	161.11	>99.9%	134.47	>99.9%	127.44	>99.9%	121.59	>99.9%	125.48	>99.9%
replace	36.25	>99.9%	34.32	>99.9%	41.44	>99.9%	31.02	>99.9%	45.48	>99.9%
sched	46.16	>99.9%	54.15	>99.9%	42.71	>99.9%	40.58	>98.8%	55.35	>99.9%
sched2	151.40	>99.9%	100.30	>99.9%	96.48	>99.9%	94.93	>99.9%	90.14	>99.9%
space	37.36	>99.9%	36.30	>99.9%	47.76	>99.9%	41.28	>99.9%	13.88	>99.9%
tcas	−19.52	>99.9%	−8.99	>99.9%	3.08	>99.9%	−5.91	>99.9%	−8.77	>99.9%
totinfo	129.15	>99.9%	125.64	>99.9%	122.23	>99.9%	94.15	>99.9%	116.56	>99.9%

The boxplots in Fig. 3 illustrate how much benefit the new approach provides for the suites individually. This figure illustrates the benefit of the VC approach over the CC and KP approaches in terms of promoting the improved APFD values for the prioritized suites. Specifically, the figure shows the APFD values by the VC, CC and KP approaches in five ranges for the given subject program has been plotted. In each plot, there are five categories of boxplots such that the boxplots of each category have the same color. In each category, the left side box represents the VC approach, the center represents the CC approach and the right side box represents the KP approach. Each category shows a specific fraction of the test suite execution ranging from 5%, 10%, 15%, 20% and 30%, respectively. The higher the position of the boxplot, the faster the prioritization technique reveals faults. It can be observed from the plots in Fig. 3 that in all cases except for *tcas*, the VC approach has considerable improvements over the CC and KP approaches in faster fault detection for various fractions of the test suite execution.

The VC approach is only degraded as compared to the CC approach for the *sched* and *sched2* programs in the range 20%. The overall results from Table 7 imply a greater likelihood of exposing faults earlier by the VC approach and show that the VC has the best performance among the three introduced history-based approaches. It is worth mentioning that the average APFD results using the VC and CC approaches and the dispersion of data for the VC and KP approaches are nearly identical for the *tcas* program.

The following additional observations can be made from the boxplots of each program except for the *tcas*. The upper quartile of the VC boxplots is much smaller than those of the CC and KP approaches. Further, the lower quartile is smaller and has a higher position in half of the programs. In addition, the median of the APFD values in the VC boxplots has a considerable higher position as compared to other approaches. All these suggest that the cases in which the VC approach may lead to an improvement over the CC and KP are significantly more than when the results are degraded. Moreover, these results are consistent with earlier average APFD and computed z values found when using the VC approach.

5.4. Experiment III: prioritizing using VC, CC and KP approaches on different coverage adequate test suites

In the above experiments, the performance of the VC approach was evaluated on the branch coverage adequate test suites. However, different coverage criteria lead to various suite sizes. In particular, when the criterion is more fine-grained, suite size grows. It is intuitive to suppose that greater suite size will give higher opportunities to the prioritization technique to put more effective test cases earlier in the list. Therefore, it is interesting to investigate and compare the effectiveness of the new approach over the CC and KP approaches on the suites with different coverage adequacy. This intuition is analyzed in this subsection.

Given the above intuition, we conducted an experiment to analyze the idea. The purpose of this experiment is to investigate the effect of coverage adequacy of a test suite on the effectiveness of the presented prioritization approaches. To achieve this, we applied the introduced history-based approaches to prioritize three categories of test suites. The first category contains 1000 branch coverage adequate test suites as used in the previous experiments. The second category includes 1000 test suites that are coverage adequate with respect to definition-use (DU) coverage criterion. Note that these suites were generated from the provided test pools for each program similar to the branch coverage adequate test suites. The suites are definition-use coverage adequate but not in a minimal manner. The third category consists of test suites that were generated randomly to have a same number of test cases as in their branch coverage counterparts. Note that for the *space* program, there is no test suite in the second category and the random test suites of the third category were generated to have the same number of test cases as in their minimal branch coverage counterparts.

Similar to the prior experiments, the prioritization has been done on 29 multi-fault versions of each subject program to be representative of 29 consecutive regression test sessions. The prioritization techniques we used are the KP, CC and VC approaches. In the experiments, P_0 for the CC and VC approaches is considered to be percentage of branch coverage of each test case. Further, it is supposed that the time and resource constraints in each regression test session allow us to execute only 30% of the test suite. Two of the approaches (i.e. KP and CC) need to set their coefficients that can be applied for prioritization. In order to make the results comparable and have a unified experiment setup, the coefficients were selected as in the previous experiment. The α coefficient for the KP approach was set to 0.5 to provide a reasonable balance in the

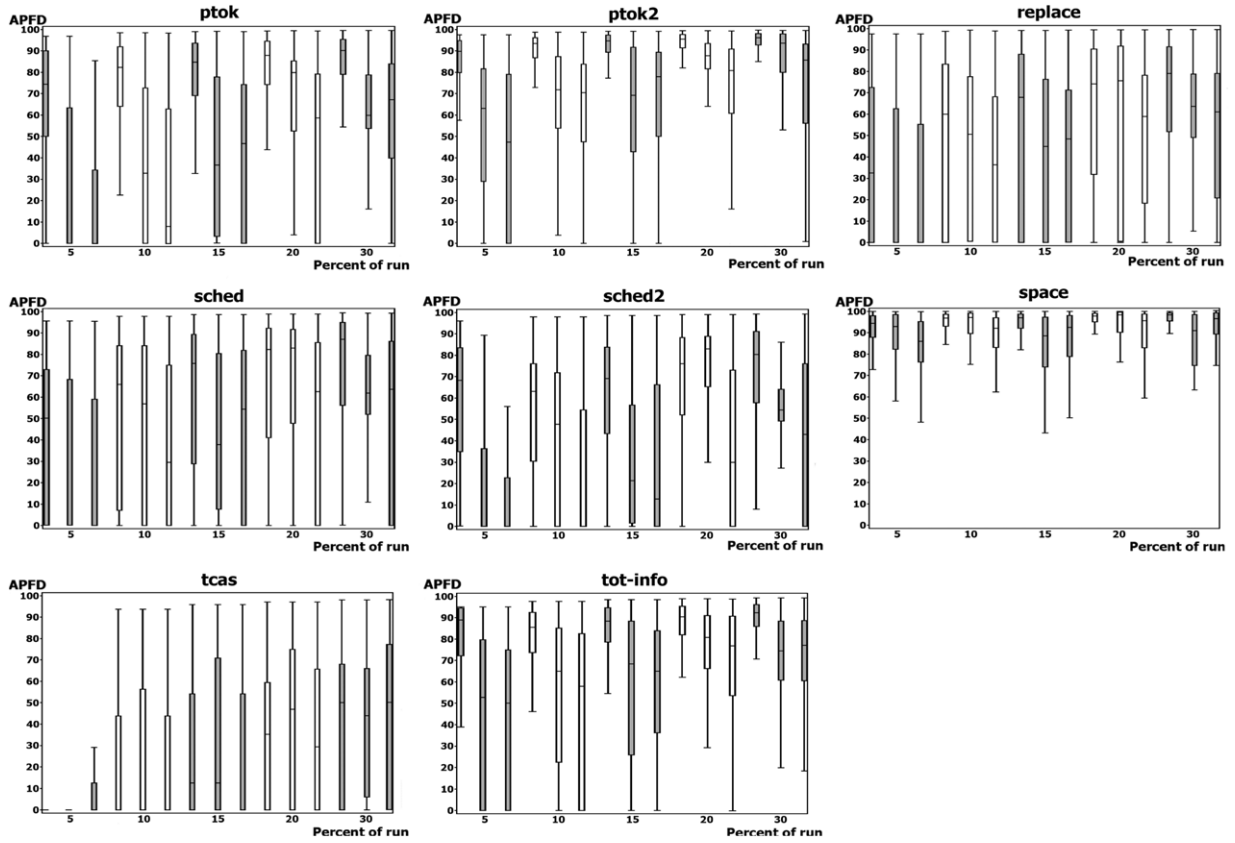

Fig. 3. The benefit of the VC approach over the CC and KP approaches in the second experiment. Each 3-grouping of boxplots in the same color correspond to VC, CC and KP approaches from left to right.

prioritization equation. Moreover, the coefficients of the equation for the CC approach have been set to 0.1, 0.5 and 0.5, respectively (suggested by the experiments in [15]).

Table 10 shows the results of conducting the above experiment. In this table, the average APFD values of the prioritizing suites with respect to the three different coverage criteria using the VC, CC and KP approaches are presented. Moreover, the average suite size of each subject program with respect to each coverage criterion among all its suites is listed in Table 10.

The following observations are made from this table. The VC prioritized test suites consistently result in considerable improved average APFD values as compared to those of the CC and KP approaches counterparts for all programs and for all coverage criteria except for branch coverage adequate test suites of the *tcas* program, in which the results were slightly degraded. The reason that the *tcas* program leads to improvement for random and definition-use coverage test suites and degrades for branch coverage suites is that it is a simple program with few numbers of branches, and the number of branches is less than the definition-use pairs. In addition, the average number of test cases in the definition-use suites is less than in the other two categories of suites. Thus, many test cases in branch coverage suites are likely to have the same percentage of branch coverage. Therefore, many test cases have the same value of P_0. As a result, the prioritization equation cannot determine the relative importance of the test cases precisely in the first session and this propagates to the subsequent test sessions. Note that the average APFD values for the branch coverage and the random test suites are greater than the average APFD values for the definition-use coverage simply because the branch coverage and random suites have a larger amount of test cases than the definition-use suites.

For all programs except for the *tcas* and *totinfo*, the average APFD values using the VC approach for the definition-use coverage test suites was improved (between 2% and 22%) as compared to their branch coverage and random test suites counterparts. This can be justified considering that the definition-use coverage is more fine-grained than the branch coverage. Hence, many test cases will have different values of P_0. This consequently causes the approach to establish an appropriate ordering in the first session, which will then propagate in the subsequent test sessions, yielding higher APFD values.

For all programs except for the *tcas* and *replace*, the VC prioritized test suites having branch coverage adequacy result in better average APFD values than random test suites. Remember that test cases in random suites were selected randomly such that the same number of test cases in their branch coverage suite counterparts was achieved.

Overall, the results in Table 10 suggest that the VC approach has a considerable improvement over the CC and KP approaches in almost all cases when applied to the coverage adequate test suites with respect to different criteria and with respect to randomly generated test suites.

Table 10
The average suite sizes for each coverage criterion and the average APFD values for each subject program when prioritizing according to the VC, CC and KP approaches.

Program name	Coverage criterion	Suite size	Approach	Test suite adequacy		
				Branch	Definition-use (DU)	Random
ptok	Branch	318.24	VC	83.39	92.27	73.62
	DU	2404.88	CC	60.46	80.38	53.79
	Random	318.24	KP	58.80	76.86	53.26
ptok2	Branch	389.44	VC	93.73	95.36	90.67
	DU	1428.20	CC	88.02	92.14	85.45
	Random	389.44	KP	72.43	74.03	80.84
replace	Branch	397.97	VC	64.39	86.46	68.52
	DU	4267.15	CC	57.61	70.92	55.35
	Random	397.97	KP	51.38	69.75	55.77
sched	Branch	224.31	VC	69.03	84.73	67.06
	DU	1800.33	CC	60.83	84.38	55.77
	Random	224.31	KP	52.17	73.43	59.37
sched2	Branch	233.64	VC	68.09	83.71	59.23
	DU	2123.21	CC	55.03	73.75	51.47
	Random	233.64	KP	43.07	52.11	52.60
space	Branch	4362.35	VC	91.70	–	74.07
	DU	–	CC	80.46	–	66.14
	Random	155.77	KP	89.44	–	68.12
tcas	Branch	83.19	VC	42.14	33.94	45.24
	DU	57.19	CC	43.01	27.87	38.54
	Random	83.19	KP	44.59	27.94	40.88
totinfo	Branch	198.05	VC	89.06	85.23	85.83
	DU	230.99	CC	73.65	71.08	69.57
	Random	198.05	KP	71.88	69.43	76.78

Tables 11 and 12 show the results of conducting a *z test* for determining whether the improvement in the average APFD values when using the VC approach over the CC and the VC approach over KP approach is statistically significant. From Table 11 it can be observed that in all cases where the average APFD value was improved when using the VC over CC, the amount of improvement was statistically significant such that the null hypothesis can be rejected with over 99.9% of confidence in all suites except for the branch coverage adequate suites of *tcas*, in which the results were slightly degraded. For the suites with the definition-use coverage adequacy of the *sched* program, the amount of improvement is also statistically significant; we can reject the null hypothesis with at least 93% of confidence.

As shown in Table 12, the difference in the APFD values when using the VC as opposed to the KP is statistically significant for all types of suites except for the branch coverage adequate suites of the *tcas* program in which the results were degraded. There is greater than 99.9% confidence for rejecting the null hypothesis.

In order to complement with the results in Tables 10–12, the boxplots in Fig. 4 are presented. These boxplots help to gain an insight into how well the VC approach performs in prioritizing test suites with different coverage adequacy. This figure illustrates the amount of benefit provided by the VC approach over the CC and KP approaches in terms of promoting improved APFD values for prioritized test suites. Specifically, for each plot, the results of the APFD values by the VC, CC, and KP approaches for the three types of test suites and for the given subject program has been plotted. In each plot, there are three categories of boxplots such that all boxplots of each category are in the same color. In each category, the left boxplot represents the VC approach, the center boxplot represents the CC approach and the right boxplot represents the KP approach. Each category shows the APFD results of prioritizing a certain type of suites.

It is obvious in the plots of Fig. 4 that for all suites and nearly for all programs, the VC approach gives considerable improvements over the CC and KP approaches due to greater median of APFD values which implies earlier detection of faults. The major exception is, as expected, the branch coverage adequate suites of the *tcas* program. Further, the height of the middle 50% of the VC boxplots is smaller than their CC and KP counterparts, which imply the stability of the VC approach in prioritizing suites. Additionally, the height of the upper quartile of VC boxplots made smaller for most of the programs and most of the suites as compared to those of the CC and KP approaches. Finally, the lower quartile of VC boxplots usually has a higher position, starting at much more than zero, and has a smaller height than their CC and KP approaches counterparts, which therefore imply a greater likelihood of exposing faults and its higher applicability in practice. All this gives us the strong confidence to justify that the VC approach improves the other two approaches. In addition, these results are consistent with earlier average APFD and computed z values when using the VC approach.

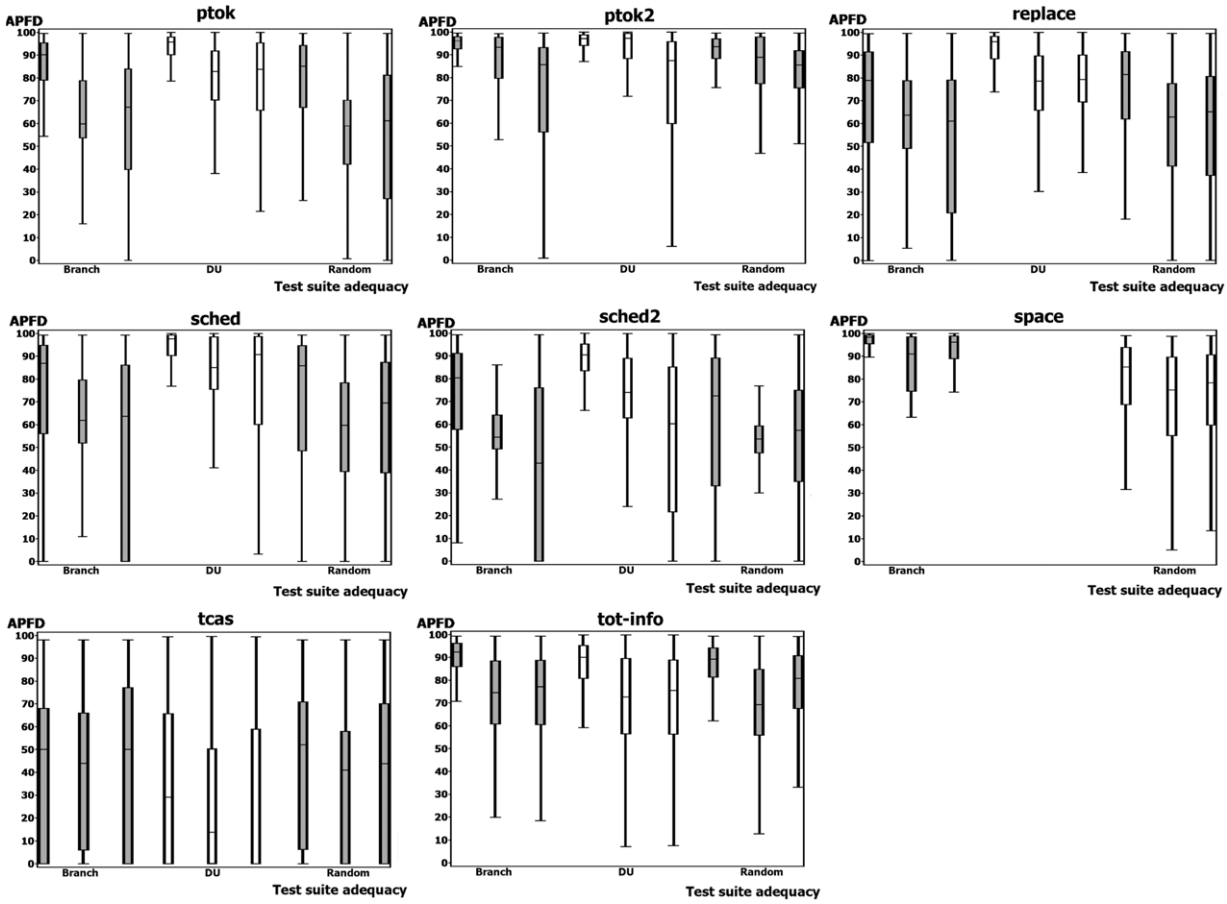

Fig. 4. The amount of benefit provided using the VC approach over the CC and KP approaches in the third experiment. Each 3-grouping of boxplots in the same color correspond to VC, CC and KP approaches from left to right.

So far, the conducted experiments have investigated the effect of test suite adequacy, time constraints and fraction of test suite execution, and the variation of coefficients to the introduced history-based techniques. These experiments evaluated the performance of each technique and compared the results together to determine the most effective approach. The results justified our intuitions and showed the generally higher improvement of the VC approach over its predecessors for most cases. In order to gain more insight into this approach, we evaluate its most important underlying parameter, P_0. For the VC approach, the prioritization equation, P_0 was defined to be the percentage of code coverage of the test case with respect to n different coverage criteria divided by n.

The usage of more than one criterion is meant to break ties among the test cases in the ordered list of the first regression test session, since the percentage of the exercised requirements by test cases may differ with respect to different criteria. For example, two test cases with the same percentage of branch coverage may exercise different percentage of definition-use pairs. We believe that using a greater number of distinguished test cases in terms of percentage code coverage will help history-based prioritization approaches to determine the relative importance of test cases in next test sessions more precisely. The next experiment is aimed at evaluating this intuition.

5.5. Experiment IV: the effect of using multiple coverage criteria for prioritizing test cases in the first session

To investigate the mentioned intuition, we conducted an experiment to see whether using multiple coverage criteria improves the average APFD values using the VC approach. To achieve this, similar to previous experiments, 1000 branch coverage adequate test suites were used to be prioritized using the proposed approach. As before, only 30% of the prioritized test suites with highest selection probabilities was selected to execute. In order to simulate consecutive regression testing process, 29 multi-fault versions of each subject program were used. For P_0, we required the percentage of code coverage of each test case with respect to multiple criteria. We used three coverage criteria: function-entry coverage and branch coverage, which are control-flow coverage criteria, and all-uses coverage, which is a data-flow coverage criterion. These coverage criteria differ in their granularity. Function-entry coverage takes into account the entrance of program functions and is considered as a coarse-grained coverage criterion. Branch coverage takes into consideration the branches of each

Table 11
The computed z values and the corresponding confidence with which the null hypothesis can be rejected when comparing the APFD values of coverage adequate suites with respect to different criteria prioritized using VC and CC.

Program name	Branch		Definition use		Random	
	z	%conf	z	%conf	z	%conf
ptok	119.55	>99.9%	109.53	>99.9%	82.92	>99.9%
ptok2	66.45	>99.9%	44.84	>99.9%	50.88	>99.9%
replace	24.46	>99.9%	69.59	>99.9%	49.03	>99.9%
sched	30.64	>99.9%	1.82	>93.1%	39.50	>99.9%
sched2	58.83	>99.9%	56.85	>99.9%	31.77	>99.9%
space	57.08	>99.9%	–	>99.9%	31.55	>99.9%
tcas	−3.31	>99.9%	22.19	>99.9%	25.51	>99.9%
totinfo	123.36	>99.9%	92.40	>99.9%	120.72	>99.9%

Table 12
The computed z values and the corresponding confidence with which the null hypothesis can be rejected when comparing the APFD values of coverage adequate suites with respect to different criteria prioritized using VC and KP.

Program name	Branch		Definition-use		Random	
	z	%conf	z	%conf	z	%conf
ptok	115.46	>99.9%	104.59	>99.9%	77.99	>99.9%
ptok2	125.48	>99.9%	121.05	>99.9%	85.88	>99.9%
replace	45.48	>99.9%	70.83	>99.9%	47.36	>99.9%
sched	55.35	>99.9%	41.87	>99.9%	26.26	>99.9%
sched2	90.14	>99.9%	128.72	>99.9%	24.75	>99.9%
space	13.88	>99.9%	–	>99.9%	23.83	>99.9%
tcas	−8.77	>99.9%	21.22	>99.9%	16.03	>99.9%
totinfo	116.57	>99.9%	91.85	>99.9%	68.16	>99.9%

program or edges in the program's control flow graph. It is considered to be a fine-grained criterion. All-uses coverage is more fine-grained (stronger) than the branch coverage. The all-uses coverage criterion [29] is the same as all definition-use pair coverage criterion, with one difference: for predicate uses,[5] a third parameter describes the destination basic block next executed as a result of the predicate's value. Thus, there may be two uses of the same variable in a given predicate: one for the predicate evaluating to true, and one for the predicate evaluating to false. Thus, the all-uses criterion is more fine-grained than the all-definition-use pair criterion.

The experiment was conducted as follows. First, the branch coverage adequate test suites were prioritized using the VC approach, with P_0 set to the percentage of function-entry coverage of each test case. Second, the experiment repeated with P_0 set to the sum of percentage code coverage of function-entry and branch coverage criteria of each test case divided by two. And finally, the experiment was performed again, in which P_0 for each test case was set to the sum of percentage code coverage with respect to the three coverage criteria (i.e. function-entry, branch, and all-uses coverage criteria) divided by three.

Table 13 shows the results of conducting the aforementioned experiment. In this table, the average APFD values of prioritizing suites using the VC approach when considering percentage code coverage for P_0 with respect to multiple criteria are presented. The results of the branch coverage (branch only column in Table 13) for P_0, were adapted from the previous experiment for comparison. In addition, the average suite size of each subject program among all its suites is listed in this table. The following observations are made from the results in Table 13. The average APFD values for prioritized suites of the programs *ptok*, *ptok2*, *sched*, *sched2*, and *totinfo* using the VC approach with P_0 set to percentage branch coverage (branch only column) has considerable improvement comparing to P_0 set to percentage function-entry coverage. In this situation, the average APFD values for the program *replace* showed slight improvement. Moreover, the results for the case of *space* programs were slightly and for the *tcas* program, considerably degraded.

As expected, when the sum of the percentage code coverage with respect to the couple criteria of branch coverage and function-entry coverage rather than only single coverage criterion has been considered for P_0, the average APFD values of the prioritized suites using the VC approach improved. For the case of function-entry coverage and branch coverage together, as compared to the function-entry coverage criterion, the amount of improvements for the *space*, *replace* and *tcas* programs is 0.24, 1.12, and 1.93 respectively. Additionally, for the function-entry coverage and branch coverage together, as compared

[5] Each use of a variable is classified into two categories. If the variable is used to compute a value to define other variables or as an output value, then it is a computational use. Otherwise, it is used to determine whether a predicate is true or false to determine the execution paths. Then, it is called a predicate use.

Table 13
The average suite sizes and the average APFD values when using the percentage of code coverage for P_0 with respect to various coverage criteria for each subject program and prioritizing according to the VC approach.

Program name	Suite size	P_0			
		Function entry	Branch	Function-entry + branch	Function-entry + branch + all-uses
ptok	318.24	75.57	83.39	82.68	84.15
ptok2	389.44	84.95	93.73	94.44	94.42
replace	397.97	64.00	64.39	65.12	66.00
sched	224.31	61.43	69.03	68.92	71.36
sched2	233.64	41.06	68.09	68.40	71.32
space	4362.35	93.67	91.70	93.91	93.95
tcas	83.19	55.40	42.14	57.33	59.45
totinfo	198.05	86.51	89.06	89.72	89.89

Table 14
The computed z values and the corresponding confidence with which the null hypothesis can be rejected when comparing the APFD values of prioritized suites using VC approach with different numbers of coverage criteria used for P_0.

Program name	P_0			
	Function-entry + branch over function-entry		Function-entry + branch + all-uses over function-entry + branch	
	z	%conf	z	%conf
ptok	36.67	>99.9%	8.66	>99.9%
ptok2	113.24	>99.9%	−0.22	<80.0%
replace	4.06	>99.9%	2.97	>99.7%
sched	25.52	>99.9%	8.21	>99.9%
sched2	98.12	>99.9%	10.89	>99.9%
space	1.59	>88.8%	0.28	<80.0%
tcas	6.51	>99.9%	7.08	>99.9%
totinfo	33.36	>99.9%	2.02	>95.6%

to the branch coverage as the single coverage criterion, the results also showed improvements, except for the *ptok* and *sched* programs in which the results has slightly degraded. Overall, the results of this experiment imply that using a couple of coverage criteria for P_0, leads to determine the relative importance of test cases at consecutive regression test sessions more precisely which yields to earlier detection of faults and noticeable improvement in APFD values. Even if one of the criteria is coarse-grained, there is a great likelihood of exposing faults faster.

The average APFD values of suites prioritized using the VC approach was improved when the sum of the three coverage criteria were used for P_0 over using a couple of coverage criteria for P_0 in all cases except for the program *ptok2* in which the results was very slightly degraded.

Table 14 shows the results of conducting a *z test* for determining whether the improvement in the average APFD values when using the VC approach with P_0 set to the sum of percentage branch coverage and function-entry coverage over the VC with P_0 set to just function-entry coverage is statistically significant. From this table, it can be seen that in all cases where the average APFD value was improved, the amount of the improvement is statistically significant such that we have at least greater than 99.9% confidence to reject the null hypothesis. For the *space* program, there is at least 88% confidence to reject the null hypothesis.

In addition, Table 14 contains the results of conducting a *z test* for determining whether the improvement in the average APFD values is statistically significant when P_0 of the VC approach is set to the sum of the percentage branch coverage, function-entry coverage, and all-uses coverage, as compared to the VC with P_0 set to the sum of function-entry and branch coverage. The results show that the amount of improvement for the programs *ptok*, *replace*, *sched*, *sched2*, *tcas*, and *totinfo* is statistically significant with over at least 95% confidence. Besides, we see that the slight average APFD degradation and improvement witnessed in the cases of *ptok2* and *space*, respectively, is not statistically significant.

The reason that VC is able to show near-consistent average improvement in nearly all programs when the number of coverage criteria used for P_0 increases is because the amount of exercised requirements (code coverage) is a suitable estimate of test case's effectiveness in terms of their ability in fault detection. When a single coverage criterion is used for the first time ordering of test cases, there may be many test cases with the same percentage of code coverage. As a result, the prioritization equation is not able to accurately distinguish essential and effective test cases in fault detection from ineffective or less effective ones. Using a second criterion, especially if it is more fine-grained, will break some of the ties among test cases with a same set of exercised requirements. Again, using the third criterion will lead to breaking even more ties yet remained.

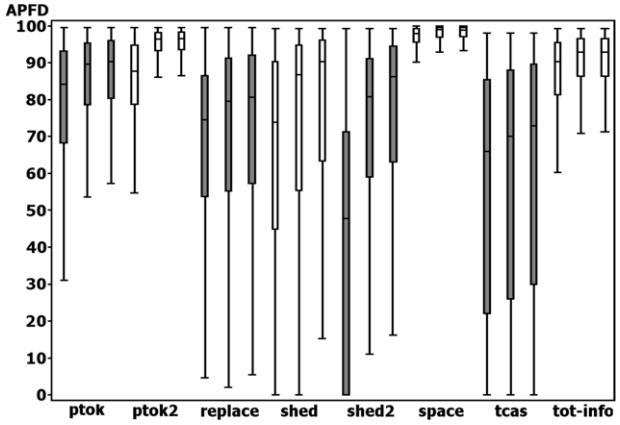

Fig. 5. The benefit of using three coverage criteria for P_0 over using only two or one coverage criteria. Each 3-grouping boxplots correspond to function-entry, function-entry + branch, and function-entry + branch + all-uses from left to right.

It was expected that using the percentage of code coverage will help history-based prioritization techniques to have considerable improvements in the average APFD values. The results of experiments in [15] and the experiments in this paper confirm our intuition. Since history-based prioritization techniques are solely based on historical performance information of each test case, any additional data about the behavior of test cases on program's code would enhance the prioritization technique to effectively schedule test cases and put those with higher likelihood of fault detection earlier in the list. The fact that the obtained results show near-consistent average improvement when using more coverage criteria for P_0 is promising for the potential of our intuition in improving the effectiveness of history-based prioritization techniques in faster fault detection. It also shows that there is something to be gained by incorporating the coverage information of test cases along with historical test case performance data in all previous test sessions during test case prioritization.

Fig. 5 provides insight into how much benefit using multiple coverage criteria for P_0 when prioritizing with the VC approach provides for prioritized suites. This figure illustrates the benefit of using the three coverage criteria for P_0 over using two or one coverage criteria in terms improving APFD values for prioritized suites.

Specifically, the results of the APFD values for the given subject program have been plotted. For each program, there are three boxplots where the left side boxplot represents the VC approach with P_0 set to percentage code coverage with respect to function-entry coverage criterion, the center boxplot represents the VC approach with P_0 set to the sum of percentage code coverage with respect to function-entry and branch coverage criteria, and the right side boxplot represents the VC approach with P_0 set to the sum of the percentage of code coverage with respect to the function-entry, branch, and all-uses coverage criteria.

The boxplots in Fig. 5 show that when going from a single coverage criterion for P_0 to a pair of coverage criteria using the VC approach, there is always considerable improvement in terms of faster detection of faults. The results are furthermore more stable since in most cases the bottom of the lower quartile is much higher and the middle 50% box is smaller. In addition, when going from a pair of coverage criteria for P_0 to the three coverage criteria using the VC approach, in most cases, there is improvement in terms of faster detection of faults and in most cases the bottom of the lower quartile is higher. All these results are consistent with earlier average APFD and computed z values when using the VC approach. Overall, the obtained results suggest that using more than one coverage criterion for P_0 is indeed useful in effectiveness of history-based prioritization techniques. In addition, when the granularity of the added coverage criterion used for P_0 is much finer, there will be a great improvement in the average APFD values of the VC approach. Finally, the number of cases in which increasing the number of coverage criteria used for P_0 in the VC approach leads to an improvement is significantly more than the number of cases when the results are unchanged or degraded.

5.6. Discussion

History-based test case prioritization has been the specific focus of this paper. However, other non history-based approaches in the literature often use white-box techniques to prioritize test cases. These techniques perform some analysis on the software code for each test case in each session of regression testing. The analysis commonly involves measuring the extent of code coverage (total or additional) of each test case from a previous version of software [4,11], the number of modifications traversed by each test case [30], estimating the fault exposing potential of test cases [11] or utilizing the number of output-influencing or potential output-influencing information of test cases [20,21]. However, such kinds of analysis necessitate a great deal of testing time and effort, often more than our use of multiple coverage criteria, especially in real applications.

In order to improve the efficacy of the proposed history-based approach, we provided some white-box information just at the first session. By this enhancement to the proposed approach, it would be considered as a hybrid approach. But, it is more

Table 15
The number of testing requirements with respect to three coverage criteria for each program obtained through instrumentation of the program code.

Program name	Function-entry	Branch	All-uses
ptok	18	137	366
ptok2	19	142	350
replace	21	140	825
sched	18	68	203
sched2	16	79	188
space	139	1116	5424
tcas	9	19	83
totinfo	10	70	290

black-box rather than white-box. Providing the code coverage information at the first session establishes an appropriate tradeoff between expensive computations of other non history-based approaches and low effectiveness of pure history-based method.

The benefit of the equation proposed in this paper can be explained by a closer comparison of those of KP and CC approaches and the equation in question. The new equation makes the use of all available historical information of test cases together for prioritizing test cases, while the KP only approach applies a single type of such information. In addition, the KP approach uses this information in a Boolean manner, whereas cumulative values are used in the new equation. The inverse coefficients are used in the KP approach. Therefore, increasing one coefficient may weaken the other parts of the equation. Our previous approach has used independent coefficients to deal with this problem. However, they are constant values. In order to adjust the effect of different parts of the equation accurately, variable coefficients in the new equation are introduced.

A possible way to make an even extreme improvement to the proposed approach is to consider the white-box information of test cases at each test session along with historical information. History-based and non history-based approaches can thus be considered to be orthogonal to each other. The use of while-box information may increase the likelihood of identifying the best test cases at each regression test session. On the other hand, this may significantly increase the computation time.

It is encouraging that the experimental results generally show significant improvement when using the VC approach, despite the fact that the *Siemens* faults are relatively difficult to detect on average. However, it is true that the results for the *tcas* program are slightly degraded. However, analyzing this program can explain why the results are as they are. The *tcas* program is the simplest of the *Siemens* suite with no loops and the fewest lines of code. This program takes twelve integer numbers as input and produces either a static error message or three possible outputs. In addition, it has no noticeable faulty version that is detected by test cases in most situations. As a result, many of test cases that exercise different execution paths within the program may still map the test cases to a same output. Therefore, *tcas* has much potential (compared with programs with wide or infinite range of output) that traversing a fault by a test case may lead to the same correct output. Moreover, as shown in Table 15, this program has the fewest branches of all the considered programs. Since the percentage of branch coverage is used for P_0 in the prioritization equation, many test cases happen to have a same value in the first session, which will then propagate to the next sessions. Thus, for the case of this program, our approach has less potential to discriminate test cases precisely and this is why the average APFD values are degraded as compared to the KP approach. Note that the amount of degradation is small indeed in most cases.

The effectiveness of our proposed approach, like other existing approaches, is dependent upon the semantics of the particular programs used. For the case of our approach, it is also dependent upon the coverage criterion used for prioritizing in the first session. Table 15 shows the number of testing requirements with respect to the three different coverage criteria for the programs used in our experiments.

The more fine-grained is the coverage criterion, the more the number of testing requirements with respect to that criterion. Thus, programs that are computationally-intensive, in which there are a lot of manipulations on variables, may have greater potential to show benefit using our approach if the all-uses, definition-use pairs or other fine-grained coverage criteria are used for P_0 in the prioritization equation. The key idea behind our history-based approach is to discriminate test cases in the first test session where there is no historical data. This can be well done using for instance an appropriate coverage criterion.

In theory, an arbitrary number of coverage criteria can be utilized for computing P_0 in the proposed equation. But it is obvious that in practice, providing the coverage information with respect to several test coverage criteria may require larger amount of testing time and resources, even though it can usually be automated [34]. This consequently will have an adverse effect on optimizing the goals of regression testing. We believe that using two different coverage criteria in computation of P_0 in the first session (e.g. a data flow based criterion and a control flow based criterion), especially if the second is more fine-grained can establish an appropriate tradeoff between the time and effort needed to gather the coverage data and faster fault detection. The results of studies by Jeffrey and Gupta [34] and our experiments also imply this intuition and give us a strong confidence to justify using a pair of coverage criteria in practical situations.

Moreover, during regression test sessions, the software code may undergo some changes. This may change the coverage information of each test case. Thus, it may be required to recompute this information at some test sessions. But, this will also defeat the purpose of regression testing optimization, which is to avoid re-running the unnecessary test cases. To address this issue, it has been proposed [35] that the coverage data from a previous version can be a reasonable estimate of the current coverage data. In other words, coverage information does not vary that much across programs modifications.

5.7. Threats to validity

In this section, we describe the potential threats to the validity of our study and how we can limit these threats.

- *Threats to construct validity*: In our studies, the measurements of the rate of fault detection and APFD values are accurate. But, APFD is not the only metric for measuring the rate of fault detection. Recently, the $APFD_c$ metric has been proposed [9], which incorporates test costs and fault severities that are not considered by APFD.
- *Threats to internal validity*: The most important internal threat to our study is instrumentation of programs' source code to compute the coverage information. To this end, we have used ATAC tool to measure the all-uses and the function-entry coverage, but the branch coverage of test cases obtained by hand-instrumentation of the source code. To ensure the correctness of instrumentation, the branch coverage of the test cases were verified. Another issue is the composition of the test suites. To limit this threat, we have used test suites from [31], which were used in the previous studies [6,11].
- *Threats to external validity*: In our study, the most important threat is whether the subject programs are representative of real programs. The *Siemens* programs are small (average 350 LOC) and their faults are hand-seeded. *The Space* program is a real application and much larger program (about 11KLOC), but it is the only one of such programs we have used. The faulty versions in *Siemens* suite are single fault versions, but we need multi-fault versions. Thus, we used faulty versions with random number of non-interfering faults. However, in practice, there may be different patterns of fault occurrences. To deal with these, we used coverage-adequate test suites with respect to the three coverage criteria (i.e. branch, function-entry and all-uses). Furthermore, the suites were created to have enough redundancy. In addition to the *Siemens* subject programs, experiments with *Space* program, which is a real program, served us as proof of concept for real applications.

6. Conclusions

In this paper, we have extended and enhanced our previous approach for test case prioritization. Contrary to the previous approach, the coefficients of the modified prioritization equation are obtained through the computation using test case historical performance data. Our idea is based on the intuition that each part of the prioritization equation should have specific effects on the test case priority and these effects may change over continuous regression test sessions. In order to evaluate our intuition, we have conducted an experimental study to compare the effectiveness of the three different history-based approaches including the proposed approach, our previous approach and that of Kim and Porter, in different situations. In our studies, the results of the prioritized suites using our approaches outperformed their prioritized counterparts using the Kim and Porter's approach in terms of improving the rate of fault detection, as measured using the APFD metric. The results also showed that the new approach has a lot of potential in terms of determining the relative importance of test cases in suites at regression test sessions with time and resource constraints.

There are some ways to extend this work. Measuring the rate of fault detection using the $APFD_c$ metric, incorporating the cost and severity of faults in the prioritization equation, conducting experiments on more available real C programs, such as *gcc*, *ant*, *nanoxml*, *etc.*, and the empirical studies on Java benchmarks can be subjects of further studies.

Acknowledgements

Special thanks go to Dr. Dennis B. Jeffrey from the University of California, Riverside, who helped us to answer a few questions about the ATAC tool. We also thank the anonymous referees of this journal, whose comments substantially improved the quality of this paper. Finally, we are grateful to Prof. Julia Lawall, the editor of this journal, for her comprehensive comments, which significantly improved the final version of the paper.

References

[1] S. Mirarab, L. Tahvildari, A prioritization approach for software test cases based on Bayesian network, in: Proc. of FASE'07, in: Lecture Notes in Computer Science, vol. 4422, 2007, pp. 276–290.
[2] I. Burnstein, Practical Software Testing: A Process-oriented Approach, Springer-Verlag, New York, 2003.
[3] G. Rothermel, R.H. Untch, C. Chu, M.J. Harrold, Test case prioritization: an empirical study, in: Proc. of the IEEE Int. Conf. on Software Maintenance, Oxford, England, 1999, pp. 179–188.
[4] G. Rothermel, R.H. Untch, C. Chu, M.J. Harrold, Prioritizing test cases for regression testing, IEEE Transactions on Software Engineering 27 (10) (2001) 102–112.
[5] K.R. Walcott, M.L. Soffa, G.M. Kapfhammer, R.S. Roos, Time-aware test suite prioritization, in: Proc. of the ACM SIGSOFT/SIGPLAN Int'l Symp. on Software Testing and Analysis, Portland, Maine, 2006, pp. 1–12.
[6] Z. Li, M. Harman, R. Hierons, Search algorithms for regression test case prioritization, IEEE Transactions on Software Engineering 33 (2007) 225–237.

[7] J.M. Kim, A. Porter, A history-based test prioritization technique for regression testing in resource constrained environment, in: Proc. of the 24th Int'l Conf. Soft. Eng., 2002, pp. 119–129.
[8] G. Malishevsky, S. Elbaum, G. Rothermel, S. Kanduri, Selecting a cost-effective test case prioritization technique, Software Quality Control 12 (2004) 185–210.
[9] G. Malishevsky, J.R. Ruthruff, G. Rothermel, S. Elbaum, Cost-cognizant test case prioritization, Tech. Rep. TR-UNL-CSE-2006-0004, Department of Computer Science and Engineering, Nebraska, Lincoln, March 2006.
[10] H. Park, H. Ryu, J. Baik, Historical value-based approach for cost-cognizant test case prioritization to improve the effectiveness of regression testing, in: Proc. of the 2nd Int'l Conf. Secure System Integration and Reliability Improvement, 2008, pp. 39–46.
[11] S. Elbaum, A.G. Malishevsky, G. Rothermel, Test case prioritization: a family of empirical studies, IEEE Transactions on Software Engineering 28 (2) (2002) 159–182.
[12] G. Rothermel, M.J. Harrold, A safe, efficient regression test selection technique, ACM Transactions on Software Engineering 6 (2) (1997) 173–210.
[13] P.G. Frankl, G. Rothermel, K. Sayre, F.I. Vokolos, An empirical comparison of two safe regression test selection techniques, in: Proc. of the 2003 Int'l. Symp. on Empirical Soft. Eng., 2003, pp. 195–205.
[14] J.M. Kim, A. Porter, G. Rothermel, An empirical study of regression test application frequency, in: Proc. of the 22nd Int'l. Conf. on Soft. Eng., 2000, pp. 126–135.
[15] Y. Fazlalizadeh, A. Khalilian, M. Abdollahi Azgomi, S. Parsa, Incorporating historical test case performance data and resource constraints into test case prioritization, in: Proc. of the 3rd Int'l Conf. on Tests and Proofs (TAP'09), in: Lecture Notes in Computer Science, vol. 5668, 2009, pp. 43–57.
[16] W.E. Wong, J.R. Horgan, S. Londonm, H. Agrawal, A study of effective regression testing in practice, in: Proc. 8th Int'l Symp. Soft. Reliability Eng., 1997, pp. 230–238.
[17] G. Rothermel, S. Elbaum, Putting your best tests forward, IEEE Software 20 (5) (2003) 74–77.
[18] A. Srivastava, J. Thiagarajan, Effectively prioritizing tests in development environment, in: Proc. of the 2002 ACM SIGSOFT Int'l. Symp. on Soft. Testing and Analysis, Roma, Italy, 2002, pp. 97–106.
[19] B. Qu, C. Nie, B. Xu, X. Zhang, Test case prioritization for black-box testing, in: Proc. 31st Annual Int'l Computer Software and Applications Conf., vol. 1, 2007, pp. 465–474.
[20] H. Agrawal, J.R. Horgan, E.W. Krauser, S. London, Incremental regression testing, in: Proc. of the Conf. on Soft. Maintenance, 1993, pp. 348–357.
[21] T. Gyimothy, A. Beszedes, I. Forgacs, An efficient relevant slicing method for debugging, in: Proc. of the 7th European Soft. Eng. Conf., Toulouse, France, 1999, pp. 303–321.
[22] D. Jeffrey, N. Gupta, Test case prioritization using relevant slices, in: Proc. of the 30th Annual Int'l. Comp. Soft. and App. Conf., vol. 1, 2006, pp. 411–420.
[23] D. Jeffrey, N. Gupta, Experiments with test case prioritization using relevant slices, Journal of Systems and Software 81 (2008) 196–221.
[24] M.J. Harrold, R. Gupta, M.L. Soffa, A methodology for controlling the size of a test suite, ACM Transactions on Software Engineering and Methodology 2 (1993) 270–285.
[25] G. Rothermel, M.J. Harrold, J. Ostrin, C. Hong, An empirical study of the effects of minimization on the fault detection capabilities of test suites, in: Proc. of the Int'l. Conf. on Soft. Maintenance, 1998, pp. 34–43.
[26] J.R. Horgan, S.A. London, ATAC: a data flow coverage testing tool for C, in: Proc. of the Symp. on Assessment of Quality Soft. Development Tools, 1992, pp. 2–10.
[27] M.E. Delamaro, J.C. Maldonado, Proteum: a tool for the assessment of test adequacy for C programs, in: Proc. of the Conf. on Performability in Computing Systems, PCS'96, 1996, pp. 79–95.
[28] Q. Yang, J.J. Li, D.M. Weiss, A survey of coverage-based testing tools, The Computer Journal 52 (5) (2009) 589–597.
[29] H. Zhu, P.A. Hall, J.H. May, Software unit test coverage and adequacy, ACM Computing Surveys 29 (4) (1997) 366–427.
[30] S. Elbaum, A.G. Malishevsky, G. Rothermel, Prioritizing test cases for regression testing, in: Proc. of the 7th Int'l Syrup. Soft. Testing and Analysis, 2000, pp. 102–112.
[31] H. Do, S. Elbaum, G. Rothermel, Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact, Empirical Software Engineering 10 (4) (2005) 405–435.
[32] SAS 9.1.3 Documentation, SAS/GRAPH 9.1 Reference, URL: http://support.sas.com/documentation/onlinedoc/91pdf/index_913.html.
[33] J.E. Freund, Mathematical Statistics, 5th ed., Prentice-Hall, 1992.
[34] D. Jeffrey, N. Gupta, Improving fault detection capability by selectively retaining test cases during test suite reduction, IEEE Transactions on Software Engineering 33 (2) (2007) 108–123.
[35] D.S. Rosenblum, E.J. Weyuker, Using coverage information to predict the cost-effectiveness of regression testing strategies, IEEE Transactions on Software Engineering 23 (3) (1997) 146–156.
[36] S. Yoo, M. Harman, Pareto efficient multi-objective test case selection, in: Proc. of the Int'l. Symp. on Soft. Testing and Analysis, 2007, pp. 140–150.
[37] N. Nisan, T. Roughgarden, E. Tardos, V.V. Vazirani, Algorithmic Game Theory, Cambridge University Press, 2007.