An Empirical Study on the Effectiveness of Time-Aware Test Case Prioritization Techniques[*]

Dongjiang You[1,2], Zhenyu Chen[1,2,*], Baowen Xu[1], Bin Luo[1,2] and Chen Zhang[1,2]
[1]State Key Laboratory for Novel Software Technology, Nanjing University, China
[2]Software Institute, Nanjing University, China
[*]zychen@software.nju.edu.cn

ABSTRACT

Regression testing is often performed with a time budget and it does not allow executing all test cases. Test case prioritization techniques re-order test cases to increase the rate of fault detection. Several time-aware test case prioritization techniques have been proposed to satisfy a time budget. Since it is difficult to collect the time cost of each test case in some cases, a natural question is whether it is worth using such information when prioritizing test cases. In this paper, two most popular criteria: statement coverage and fault detection are considered for time-aware test case prioritization. We investigate whether the time cost of each test case affects the effectiveness of prioritization techniques, i.e. the rate of statement coverage and the rate of fault detection. Our empirical study shows that: although the techniques considering the time cost of each test case are slightly better than the techniques not considering such information in some cases, they have no significant difference in most cases.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Experimentation, Verification.

Keywords

Test case prioritization, time, statement coverage, fault detection.

[*]The work described in this article was partially supported by the National Natural Science Foundation of China (90818027, 60803007, 61003024, 61011130171), the National High Technology Research and Development Program of China (863 Program: 2009AA01Z147), the Major State Basic Research Development Program of China (973 Program: 2009CB320703).

1. INTRODUCTION

Regression testing is used to validate modified software and confirm that no new fault has been introduced into previously verified code. Typically, regression testing involves executing a large number of test cases, and it is very time-consuming. Therefore, regression testing is often performed with a time budget and it does not allow executing all test cases.

Test case prioritization problem is introduced and defined to re-order test cases such that more faults are revealed as early as possible [1]. Prioritization techniques are based on different coverage criteria such as statement coverage, block coverage, probability of exposing faults, etc. [1, 2, 3].

The time-aware test case prioritization problem was introduced by Walcott et al. in [4]. They proposed a test case prioritization technique that used a genetic algorithm in light of testing time constraints. Zhang et al. proposed time-aware test case prioritization techniques using integer linear programming [5]. A suitable subset of test cases can be selected firstly and then these test cases were re-ordered. Alspaugh et al. used knapsack solvers to prioritize test cases, such that a test suite was re-ordered rapidly to cover the test requirements and it always terminated within a specified time limit [6]. Some newly proposed test case prioritization techniques may be applied to time-aware circumstances. Ma et al. proposed test case prioritization techniques based on the analysis of software structure [7]. Yoo et al. used clustering techniques to group test cases to achieve effective and scalable prioritization [8]. Zhang et al. proposed test case prioritization techniques based on varying testing requirement priorities and test case costs [9].

The time budget is a practical requirement in regression testing. In [10, 11], Hyunsook Do et al. conducted a series of controlled experiments to study the effects of time constraints on test case prioritization. Their studies showed that the time budgets could impact the costs and benefits of test case prioritization techniques. Although the effects of regression testing time constraints have been studies deeply, the effects of time cost of each test case have not been studied. In [10, 11], the test cases were assumed to have the same time cost.

Furthermore, whether the time cost of each test case is a key factor is a non-trivial problem. First, a test case which costs long time often covers more parts of program, then it may have strong fault detection capability. Second, more test cases can be executed within a time limit if each test case costs short time. Third, it always needs extra expense to collect the time cost information of each test case. It

Table 1: Subject Programs

Subject	Lines of code	Test pool size	Number of versions	Average small test suite size	Average large test suite size	Total test case execution time (s)	Average test case execution time (s)	Longest test case execution time (s)	Shortest test case execution time (s)
print_tokens	726	4130	7	16	318	7.56268	0.00183	0.01798	0.00167
print_tokens2	570	4115	10	11	389	7.98576	0.00194	0.03678	0.00175
replace	564	5542	31	18	397	11.01607	0.00199	0.30888	0.00160
schedule	412	2650	9	8	224	5.26270	0.00199	0.01482	0.00179
schedule2	374	2710	9	7	233	5.44754	0.00201	0.00350	0.00175
tcas	173	1608	41	5	83	2.97943	0.00185	0.00648	0.00166
tot_info	565	1052	23	7	198	2.19234	0.00208	0.00306	0.00180
space	9564	13585	34	155	4362	33.05996	0.00243	0.03892	0.00186

is difficult to collect the accurate time cost information in some cases, such as manual testing. Therefore, it motivates us to investigate whether the time cost of each test case is a key factor that impacts the effectiveness of time-aware test case prioritization techniques.

This paper provides an empirical study to investigate the time-aware test case prioritization techniques. In summary, our study makes the following main contributions: although the techniques considering the time cost of each test case are slightly better than the techniques not considering such information under some circumstances, they generally have no significant difference both for the purpose of statement coverage and fault detection. That is to say, it is not worth considering the time cost of each test case for time-aware test case prioritization in most cases. For this result, we suggest using the techniques that consider only coverage or assuming all the test cases have the same time cost when prioritizing test cases.

The rest of this paper is organized as follows. Section 2 introduces the time-aware test case prioritization problem. Section 3 describes our experiment. Section 4 reports our experiment results and analysis. Section 5 analyzes the threats to validity. Section 6 presents the conclusions and future work.

2. TIME-AWARE TEST CASE PRIORITIZATION

According to [4, 5], the time-aware test case prioritization problem is defined as follows.

Time-Aware Test Case Prioritization Problem:

Given: A test suite T and the time budget $time_{max}$, the permutations of all subsets of T is denoted by PT; two functions from PT to the real numbers, f and $time$.

Problem: Find $T' \in PT$ and $time(T') \leq time_{max}$, such that

$(\forall T'')(T'' \in PT)(time(T'') \leq time_{max})[f(T') \geq f(T'')]$

In this problem, PT represents the set of all possible orderings of test cases in the subsets of T. The function $time$ measures the execution time of each PT, and we also use $time(t)$ to represent the execution time of test case t for simplicity.

The function f measures the fitness value of each PT. The fitness function is often used as an evaluation metric. A popular metric called $APFD$, was introduced by Rothermel et al. to evaluate the rate of entity coverage and fault detection of a test suite [1, 2]. Walcott et al. applied a penalize measure in $APFD$ [4]. Elbaum et al. proposed a new evaluation metric called $APFD_c$ to consider time cost and fault severity into the evaluation of prioritized test suites [12]. In this paper, the function f quantifies the rate of statement coverage and the rate of fault detection of a test suite, which will be described in detail in section 3.6. With the function f, the problem requires us to seek for an ordering that satisfies the time budget and has the highest fitness value.

3. EXPERIMENT

3.1 Research Question

We are interested in the following research question:

Both for the purpose of statement coverage and fault detection, is the time cost of each test case a key factor for time-aware test case prioritization?

3.2 Subject Programs, Test Suites, and Versions

In order to improve the generality of our results in this paper and limit the threats to our experiment's validity, the following three issues about experimental subjects were considered.

3.2.1 Subject Programs

We used eight C programs as subjects. Both large programs and small programs were used in our experiment, ranging from 374 to 9564 lines of code. All these programs and their test suites were taken from *Software-artifact Infrastructure Repository (SIR)* [13]. Table 1 shows the basic information of the eight programs.

3.2.2 Test Suites

Both large test suites and small test suites were used in our experiment. The average size of small test suites ranged from 5 to 155, and the average size of large test suites ranged from 83 to 4362. The two groups of test suites were generated in the following way [13]: in order to generate small test suites, test cases were randomly selected from the test pool and added to the test suite as long as they added coverage. This was repeated until the test suite achieved full branch coverage. In order to generate large test suites, test cases were randomly selected from the test pool and added to the test suite until full branch coverage was achieved. For each group of test suites, 1000 test suites were made. In order to reduce the time cost of our experiment, without significant

Table 2: Prioritization Techniques

No.	Abbr.	Description
1	*rand*	Randomized ordering
2	*st-tot*	Prioritizing test case on total statement coverage
3	*st-add*	Prioritizing test case on additional statement coverage
4	*rt-tot*	Prioritizing test case on the ratio of total statement coverage to time cost
5	*rt-add*	Prioritizing test case on the ratio of additional statement coverage to time cost
6	*ILP-tot*	Time-aware total statement coverage prioritization via ILP
7	*ILP-add*	Time-aware additional statement coverage prioritization via ILP

loss of generality, we randomly selected 100 test suites for each group of test suites.

3.2.3 Multi-fault Versions

Our experiment required programs with varying numbers of faults. We generated these versions in the following way [2]: each subject program was provided with a correct base version and multiple single-fault versions. We created multi-fault versions by combining these single-fault versions. In order to limit the threats to our experiment's validity, we generated the same number of versions for each of the programs. For each subject program, we created 10 multi-fault versions, the number of faults contained in each version varied randomly from 1 to the total number of single-fault versions of that program.

Note the fact that types and locations of faults may affect the results of our experiment, but this is not the scope of our discussion. In our experiment, we did not distinguish different types and locations of faults.

3.3 Test Case Prioritization Techniques

Table 2 shows the seven techniques considered in our experiment.

For all the seven techniques, if the test cases had the same priority, we prioritized them randomly.

rand was considered as an experimental control. *st-tot*, *st-add*, *rt-tot*, and *rt-add* are widely known traditional test case prioritization techniques. *st-tot* and *st-add* are techniques that only consider statement coverage, *rt-tot* and *rt-add* are techniques that consider both time cost and statement coverage, i.e. the coverage efficiency of test cases.

ILP-tot and *ILP-add* were proposed by Zhang et al. in [5], a suitable subset of test cases can be selected firstly using integer linear programming, and then these selected test cases were re-ordered using traditional test case prioritization techniques.

Among all the seven techniques, four of them (*rt-tot*, *rt-add*, *ILP-tot*, *ILP-add*) depend on the time cost information of each test case; the others (*rand*, *st-tot*, *st-add*) do not need such information.

In this paper, besides traditional techniques, we also studied ILP techniques. We did not consider other techniques such as genetic algorithms and 2-optimal algorithms [4, 14] because the performances of our selected techniques are better than the others in most cases and generally have no significant difference [5, 14, 15].

3.4 Time Budgets

We used five time budgets for each subject program: 5%, 25%, 50%, 75%, and 100% of the execution time of the entire test suite. Following [5], we used 5% of the total execution time to represent a very tight time budget. We also used 100% of the total execution time, which was the same as traditional test case prioritization, to represent no time budget for each subject program.

3.5 Effectiveness Measure

According to the effectiveness measure used in [1, 2, 4, 5, 12], and depending on the research question in this paper, i.e. both the rate of statement coverage and the rate of fault detection were studied, two evaluation metrics were used in this paper, $APSC_{TA}$ and $APFD_{TA}$.

$APSC_{TA}$ (Average Percentage Statement Coverage, Time-Aware): This measures the rate at which a prioritized test suite covers statements.

Consider a prioritized test suite T' containing n test cases that covers m statements. Let TS_j be the first test case in the prioritized order that covers statement j, and t_i be the time cost of test case i. The $APSC_{TA}$ for this order is given by the following equation:

$$APSC_{TA} = \frac{\sum_{j=1}^{m} (\sum_{i=TS_j}^{n} t_i - \frac{1}{2} t_{TS_j})}{\sum_{i=1}^{n} t_i \times m}$$

$APFD_{TA}$ (Average of the Percentage of Faults Detected, Time-Aware): This measures the rate at which a prioritized test suite detects faults.

Consider a prioritized test suite T' containing n test cases that detects m faults. Let TF_j be the first test case in the prioritized order that detects fault j, and t_i be the time cost of test case i. The $APFD_{TA}$ for this order is given by the following equation:

$$APFD_{TA} = \frac{\sum_{j=1}^{m} (\sum_{i=TF_j}^{n} t_i - \frac{1}{2} t_{TF_j})}{\sum_{i=1}^{n} t_i \times m}$$

For simplicity, we consider $APFD_{TA}$ as an example in the following of this section.

Since T' is a subset of T, it may contain fewer test cases than T. Therefore T' may not be able to detect all defects. Based on the penalize measure used in [4], if a fault j cannot be detected by any test case in T', we define

$$t_{penalize} = t_{j+1}$$

Then the part

$$(\sum_{i=TF_j}^{n} t_i - \frac{1}{2} t_{TF_j})$$

in the $APFD_{TA}$ equation will be calculated as

$$-\frac{1}{2} t_{penalize}$$

Note that when the time budget is 100%, i.e. the number of test cases in T' equals that in T, t_{n+1} is a meaningless value. At this moment, we define

$$t_{n+1} = \frac{\sum_{i=1}^{n} t_i}{n}$$

This measure would possibly cause a prioritized test suite that finds few faults to have a negative $APFD_{TA}$ value. Thus, test suites finding few faults are penalized in this way.

Note the following special case: If the time budget is so tight that no test case is selected to execute, the value of n will be 0, and $APFD_{TA}$ will be *negative infinity*. In our analysis of results, we removed such meaningless values. The range of $APFD_{TA}$ is $(-\infty, 1)$. The higher the $APFD_{TA}$ value is, the better the effectiveness of the prioritization technique detects faults.

3.6 Information Collected for Coverage and Execution Time

In our experiment, we used statement coverage as prioritization criteria. We removed the statements that could not be covered by any test case in collection of statement coverage information.

We used a test coverage tool *gcov* in Linux to collect the coverage information of test cases [16]. We developed scripts to run all the test cases for each subject program and created the profiling files of **.gcov* format. Then we interpreted all the profiling files to generate coverage matrix, in which a column stands for a statement and a row stands for a test case. If a test case could cover a statement, the corresponding position in the coverage matrix would be marked as 1, otherwise marked as 0.

In order to generate fault matrix, we ran all test cases on correct base version and faulty versions. In our experiment, if the output of a test case T_i for a single-fault version F_j is different from its output for the correct base version, we define test case T_i can detect fault F_j. In the fault matrix, a column stands for a fault and a row stands for a test case. If a test case could detect a fault, the corresponding position in the fault matrix would be marked as 1, otherwise marked as 0.

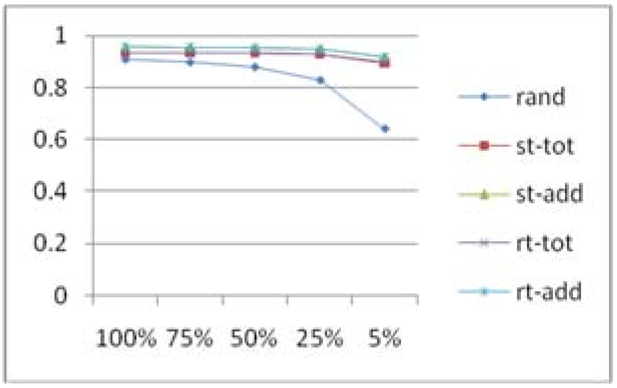
In order to obtain the execution time of each test case, we executed the whole test pool and used a tool called *time* provided by *SIR* to record the time cost of each test case. To ensure the accuracy of the time cost information, each test pool was executed 10 times and the final test case execution time was the average over them.

4. RESULTS AND ANALYSIS

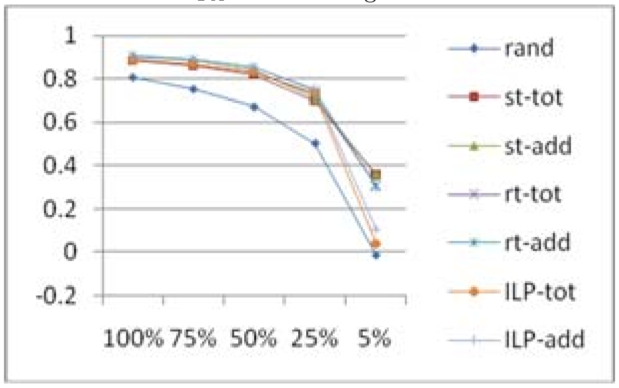
Figure 1 shows the $APSC_{TA}$ and $APFD_{TA}$ results of both large test suite and small test suite on *print_tokens*. In the line diagram, the X-axis shows the five time budgets, and the Y-axis shows the $APSC_{TA}$ or $APFD_{TA}$ values. Figure 2 shows the $APSC_{TA}$ and $APFD_{TA}$ results on *print_tokens* at 50% time budget. In the boxplot, the X-axis shows different techniques, and the Y-axis shows the $APSC_{TA}$ or $APFD_{TA}$ values. Due to the page limit, we only put figures of the first subject program here.

Note that when the test suite is large, it is very time-consuming even impossible to prioritize test cases using ILP techniques. A large number of variables make this NP-hard problem unsolvable. In [5], for their two subject programs, the total number of test cases is 53 and 209, which is much less than that in our subject programs. Therefore, for large test suite, we did not get any result of ILP methods; and for small test suite, we performed all techniques successfully.

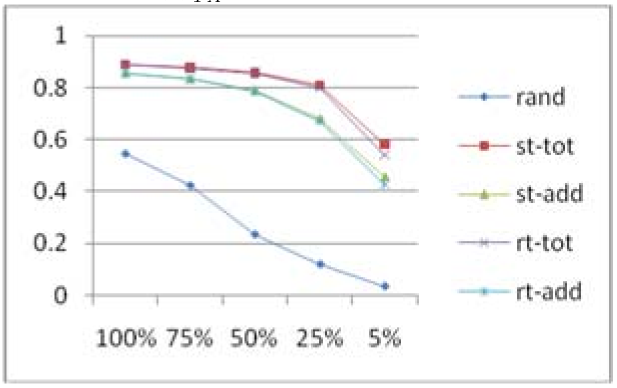
From the observation of Figure 1 and Figure 2, we can conclude that: both for the purpose of statement coverage and fault detection, and both for large test suite and



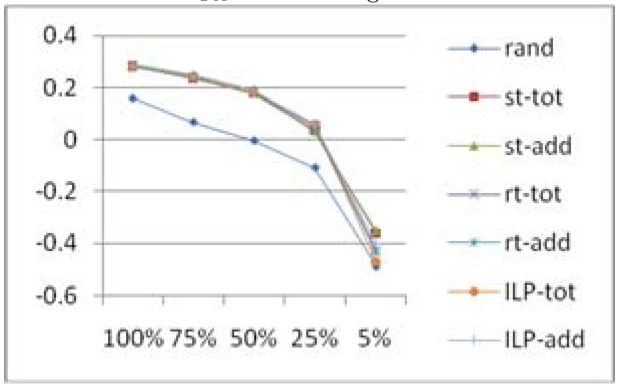
$APSC_{TA}$ values of large test suite



$APSC_{TA}$ values of small test suite

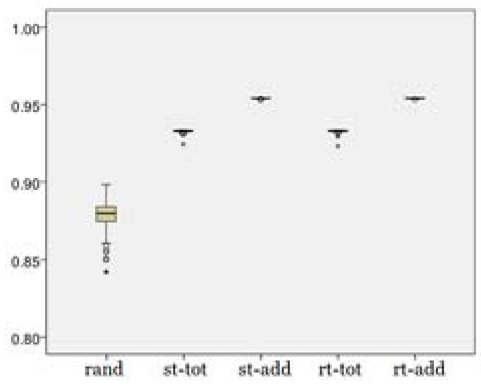


$APFD_{TA}$ values of large test suite

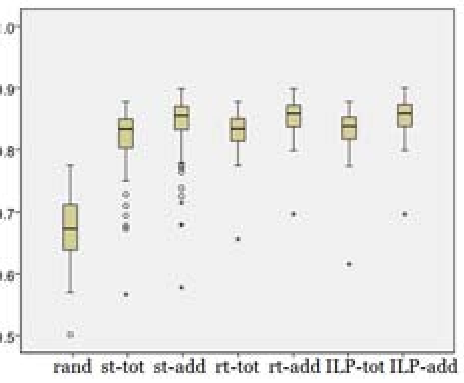


$APFD_{TA}$ values of small test suite

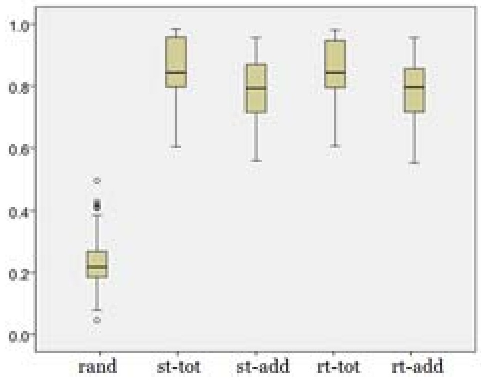
Figure 1: Line diagrams of *print_tokens*



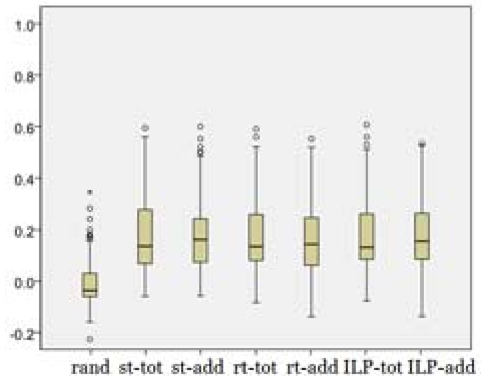
$APSC_{TA}$ values of large test suite



$APSC_{TA}$ values of small test suite



$APFD_{TA}$ values of large test suite



$APFD_{TA}$ values of small test suite

Figure 2: Boxplot of *print_tokens* at 50% time budget

small test suite, (1) the effectiveness of time-aware test case prioritization techniques is significantly better than that of randomized prioritization; (2) the effectiveness of additional techniques (*st-add*, *rt-add*, and *ILP-add* for small test suite) is very close, so is the effectiveness of total techniques (*st-tot*, *rt-tot*, and *ILP-tot* for small test suite).

We performed ANOVA (ANalysis Of VAriance) to investigate the difference among these techniques. The LSD (Least Significant Difference) method was used in multiple-comparison to compare the six techniques (*st-tot*, *st-add*, *rt-tot*, *rt-add*, *ILP-tot*, and *ILP-add*) pair wise. Table 3 shows the summary of ANOVA analysis. In *Total groups* column, the number should be 40 in all rows (i.e. 8 subject programs and 5 time budgets), but for small test suite the number is 35 because there is no test case selected for *print_tokens2*, *schedule*, *schedule2*, *tcas*, and *tot_info* at 5% time budget for all the 100 test suites.

From the observation of these tables, we can conclude that prioritizing test cases both on their coverage and time cost (*rt-add*, *ILP-add*) (*rt-tot*, *ILP-tot*) cannot significantly outperform that only on their coverage (*st-add*) (*st-tot*) in most cases both for the purpose of statement coverage and fault detection. Due to the fact that it is difficult to collect the accurate time cost information of each test case in some cases, we suggest that it is enough for prioritizing test cases only on their coverage to improve the rate of statement coverage and the rate of fault detection within the time limit. Meanwhile, we believe that assuming all the test cases have the same time cost is also a good choice [10, 11]. Under some circumstances, such as small test suite and tight time budget, *ILP* methods are efficient and significantly better than the other techniques.

5. THREATS TO VALIDITY

5.1 Internal Validity

Threats to internal validity are uncontrolled factors that are also responsible for our results. The first threat is that there are defects in our implementation of prioritization techniques and effectiveness measurement procedures. To reduce this threat, we reviewed all the implementation code before conducting our experiment. The second threat is that our measurement of time cost for each test case is not accurate. To reduce this threat, we used a tool called *time* provided by *SIR* to record time, and we ran the test case pool 10 times, and got the final time cost value as the average over them.

5.2 External Validity

Threats to external validity are the representativeness of our subject programs and experiment procedures. The first threat is that the subject programs with their test suites and faulty versions may not have generality. The second threat is that our experiment process may not be representative of real industrial testing process. To reduce these threats, we used widely used *Siemens* programs and a relatively large program space in our experiment. Furthermore, we used different test suite sizes and different time budgets to simulate the real situations.

5.3 Construct Validity

Threats to construct validity are mainly concerned with the evaluation metrics in our experiment. In order to measure the rate of statement coverage, *APSC* has been used

Table 3: ANOVA Summary

Evaluation metric	Techniques	Test suite size	Total groups	Number of groups that *st* methods have no significant difference with *rt* and *ILP* methods	Number of groups that *st* methods are significantly better than *rt* and *ILP* methods
$APSC_{TA}$	Total	Large	40	16	6
$APSC_{TA}$	Total	Small	35	30	0
$APSC_{TA}$	Additional	Large	40	39	0
$APSC_{TA}$	Additional	Small	35	30	0
$APFD_{TA}$	Total	Large	40	19	8
$APFD_{TA}$	Total	Small	35	30	0
$APFD_{TA}$	Additional	Large	40	34	5
$APFD_{TA}$	Additional	Small	35	30	0

by [14, 15], and in order to measure the rate of fault detection, $APFD$ has been widely used by [1, 2, 3, 4, 5, 6, 12]. In our experiment, we combined the idea provided by [12] with the penalize measure used by [4]. Under some particular conditions, these evaluation metrics are identical with ours.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have studied the effectiveness of some statement coverage based time-aware test case prioritization techniques. Our experiment results show that both for the purpose of statement coverage and fault detection, the time cost of each test case is not a key factor for time-aware test case prioritization. Therefore, it is not worth considering the time cost of each test case in most cases. We suggest using the techniques that consider only coverage or assuming all the test cases have the same time cost when prioritizing test cases. Under some circumstances, such as small test suite and tight time budget, techniques considering the time cost of each test case can significantly outperform traditional techniques.

We will investigate the following issues in the future. First, we will introduce some newly proposed prioritization techniques and some other coverage criteria. Second, we will investigate which factors affect the effectiveness of prioritization techniques. Third, we will conduct experiments on a wider range of subject programs.

7. REFERENCES

[1] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. Test case prioritization: An empirical study. In *ICSM*, pages 179–188, 1999.

[2] Sebastian G. Elbaum, Alexey G. Malishevsky, and Gregg Rothermel. Test case prioritization: A family of empirical studies. *IEEE Transactions on Software Engineering*, 28(2):159–182, 2002.

[3] Hyunsook Do, Gregg Rothermel, and Alex Kinneer. Empirical studies of test case prioritization in a junit testing environment. In *ISSRE*, pages 113–124, 2004.

[4] Kristen R. Walcott, Mary Lou Soffa, Gregory M. Kapfhammer, and Robert S. Roos. Time-aware test suite prioritization. In *ISSTA*, pages 1–12, 2006.

[5] Lu Zhang, Shan-Shan Hou, Chao Guo, Tao Xie, and Hong Mei. Time-aware test-case prioritization using integer linear programming. In *ISSTA*, pages 213–224, 2009.

[6] Sara Alspaug, Kristen R. Walcott, Michael Belanich, Gregory M. Kapfhammer, and Mary Lou Soffa. Efficient time-aware prioritization with knapsack solvers. In *WEASELTech*, pages 17–31, 2007.

[7] Zengkai Ma and Jianjun Zhao. Test case prioritization based on analysis of program structure. In *APSEC*, pages 471–478, 2008.

[8] Shin Yoo, Mark Harman, Paolo Tonella, and Angelo Susi. Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge. In *ISSTA*, pages 201–212, 2009.

[9] Xiaofang Zhang, Changhai Nie, Baowen Xu, and Bo Qu. Test case prioritization based on varying testing requirement priorities and test case costs. In *QSIC*, pages 15–24, 2007.

[10] Hyunsook Do, Siavash Mirarab, Ladan Tahvildari, and Gregg Rothermel. An empirical study of the effect of time constraints on the cost-benefits of regression testing. In *SIGSOFT FSE*, pages 71–82, 2008.

[11] Hyunsook Do, Siavash Mirarab, Ladan Tahvildari, and Gregg Rothermel. The effects of time constraints on test case prioritization: A series of controlled experiments. *IEEE Transactions on Software Engineering*, 36(5):593–617, 2010.

[12] Sebastian G. Elbaum, Alexey G. Malishevsky, and Gregg Rothermel. Incorporating varying test costs and fault severities into test case prioritization. In *ICSE*, pages 329–338, 2001.

[13] Hyunsook Do, Sebastian G. Elbaum, and Gregg Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering*, 10(4):405–435, 2005.

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

[15] Sihan Li, Naiwen Bian, Zhenyu Chen, Dongjiang You, and Yuchen He. A simulation study on some search algorithms for regression test case prioritization. In *QSIC*, pages 72–81, 2010.

[16] Hao Zhong, Lu Zhang, and Hong Mei. An experimental study of four typical test suite reduction techniques. *Information & Software Technology*, 50(6):534–546, 2008.