Dynamic Test Case Prioritization based on Multi-objective*

Xiaolin Wang[1, 2]
[1]School of Computer Engineering and Science, Shanghai University
[2]Shanghai Key Laboratory of Computer Software Evaluating & Testing
Shanghai 200444, China
wangxiaolin@shu.edu.cn

Hongwei Zeng[1]
[1]School of Computer Engineering and Science, Shanghai University
Shanghai 200444, China
zenghongwei@shu.edu.cn

***Abstract*—Test case prioritization technology is to sort the test cases before the software testing designed to improve test efficiency. This paper presents a dynamic test case prioritization technique based on multi-objective. It integrates several traditional single-objective technologies so that makes it more flexible. This technology, from five dimensions, calculates prioritization values of test cases separately. Then a weighted sum is made to the values and it sorts the test cases according to the values. The results return to the storage in order to dynamically adjust the sort of test cases. This technology not only meets the high demands of regression testing, but also ensures the high efficiency of the test results.**

Keywords—Test case prioritization; Multi-objective; Dynamic; Regression testing

I. INTRODUCTION

Nowadays, many test case prioritization methods have been proposed. As the survey[1] shows, most of them are focus on one objective to prioritize from single dimension, such as coverage or fault detection rate.

Rothermel et al[2-4] began to research test case prioritization(TCP) technique in 1999. Their initial research mainly focuses on the sort of fine-grained entity, such as statement coverage and transition coverage and verifies its effectiveness. Elbaum et al[3-5] studied the sorting method of coarse-grained entity, which has low test cost but the performance is lower than fine-grained entity. Hema et al[6-7] divided the test requirements into four elements and proposed the PORT algorithm. The prioritization method based on historical information is proposed by Jung-Min in 2002[8]. Nie and Xu et al[9] proposed several TCP methods based on historical information, test requirement and test cost.

However, there are many factors that restrict the test case prioritization in the actual software development. The paper takes five typical test objectives into consideration and merges the existing TCP methods from multiple dimensions. The new defined method can produce more flexible test case prioritization and meet the test requirements better.

* This work is supported by National Natural Science Foundation of China (NSFC) under grant No. 61073050 and No. 61170044.

II. TEST CASE PRIORITIZATION

The test case prioritization problem is a research hotspot in the field of software testing. It is defined as follows[4]:

Given: T, a test suite already selected, PT, the set of all possible prioritizations (orderings) of T, and f, an objective function from PT to the real numbers which applied to any such ordering, yields an award value for that ordering.

Problem: Find $T' \in PT$ such that

$$(\forall T'')(T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')].$$

Currently, the test case prioritization techniques mostly aimed at single objective, including:

Coverage Prioritization: It counts the total number of statements (or branch) covered by each test case, and then sorts the test suit according to the number. However, this strategy may make statement coverage of a test case be a subset of another test coverage, and eventually, there are still some statements which have not been covered by any test cases. Therefore, additional strategy is appeared. It first picks the test case with the greatest coverage, and then, successively adds those test cases that cover the most yet uncovered parts. When multiple test cases cover the same number of statements, the additional strategy is necessary to select one of these test cases. Generally, be random selection[2].

Fault-Exposing-Potential (FEP) Prioritization: This technique achieves by the ability to expose faults and mutation analysis is used to determine these values[2]. Each application of a mutation operator will create a mutant of the source code, which makes a single valid syntactic change. The mutation score represents the ratio of mutants that a test-suite can distinguish from the source code. The mutation score can be calculated for each test case separately, and then used as a decision value for TCP.

Requirement Property Relevance Prioritization: It sorts the test cases according to the number of requirement properties covered. Suppose the set of requirement properties $R = \{r_1, r_2, \cdots r_m\}$ and test suite $T = \{t_1, t_2, \cdots t_n\}$, for every test case $t \in T$, count the number of requirement properties which

978-1-4799-5604-3/14/$31.00 copyright 2014 IEEE

t satisfies. Then, the test cases are sorted in the light of the counting value[9].

History Prioritization: This technique prioritizes test cases based on historical execution data[10]. This method helps to reduce spending. In the long run, it can enhance the effectiveness of regression testing. For a test case, using historical information of each prioritization, increase or decrease its likelihood in the current test session. A probability value is given to the history and the value of the current session is calculated based on the number of the fault detection (or coverage). Finally, the calculated value of a test case is equal to the sum of the current number multiplied by a probability and the historical value multiplied by another probability.

Time Spending Prioritization: This method is that count execution time of each test case and then sorts the test suite according to the ascending order of the calculated value.

III. METHODOLOGY OF MULTI-OBJECTIVE PRIORITIZATION

A. Basic Idea

Currently, many prioritization techniques are mostly based on an objective to sort test suite on a single dimension. The common one is based on coverage. However, only considering one objective makes prioritization not flexible. Therefore, considering multi-dimensional prioritization model, this paper proposes a dynamic prioritization technique. Fig. 1. is a framework.

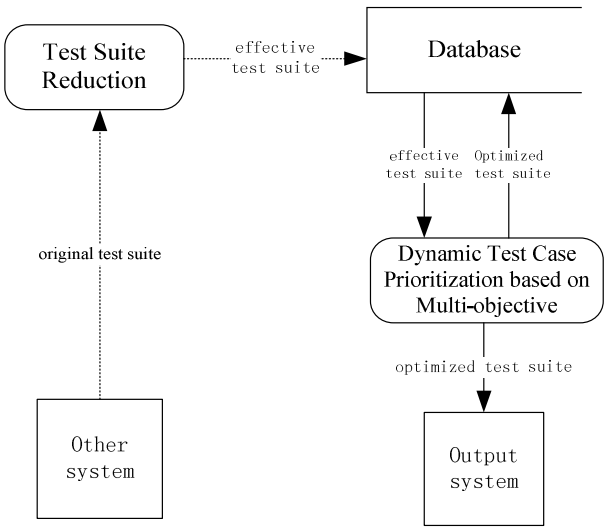

Fig. 1. framework of Dynamic Test Case Prioritization technique

The data flow of Fig. 1. is from the lower left corner. We assume that related technologies have been used to generate test suite and extract a valid set of test cases by using the test suite reduction method. Then, store it in the test database, such as the box in the upper right. Suppose that the steps in dotted lines are not our concerns in this paper. Then using dynamic test case prioritization technique based on multi-objective to sort a set of test cases which has been produced, it is a focus in the study of this article. Finally, output the well-sorted test suite and return to store in the test database for the next use.

It is need to test constantly to ensure quality of the software during the software development life cycle. So, in practice, it should maximize reuse of test cases. Not considering the added test cases due to the changing requirement, supposing the test suite always has the validity and integrity during the whole software development life cycle, sort generated for each test is stored in the test database, which as a historical information providing for the use of the next TCP. As shown in Fig. 1. the cycle on the right side represents the data flow of dynamic TCP of the entire software development cycle.

B. Prioritization Value Calculation

In prioritization technique, each test case is assigned a prioritization value *PV*. The calculation of *PV* involves five dimensions: *C*, *FEP*, *RP*, *H* and *TC*, respectively representing coverage, ratio of fault exposing, requirement property relevance, history information and time spending.

Definition 1: Given T, a set of test cases , for each $t \in T$, its prioritization value *PV* is calculated as follows:

$$PV(t) = C(t) + FEP(t) + RP(t) + H(t) + TC(t) \quad (1)$$

Five symbols represent respectively the five metrics on five dimensions. Firstly, we calculate metrics on a single dimension respectively and then sum weightly them to the final prioritization value *PV*.

(1) For the coverage, there are a lot of sophisticated techniques, and the paper selects the statement coverage. Suppose that there is a given set of test cases $T = \{t_1, t_2, \cdots t_n\}$, for each $t \in T$, count the number of statements, denoted as C(t). We use additional strategy[2], which introduces a feedback mechanism. When a test case is executed, we need to update the coverage information in the real time for the remainder test cases. When all statements covered, reset them be not covered and apply the above process for the remaining test cases iteratively. Greedy algorithm is used for implementation, as follows.

Algorithm 1 Prioritization based on statement coverage

Input:

T: a queue of orderly test cases

S: a set of statements

Output:

T': a well-ordered queue

Cover(t): the statement coverage set of t

```
1 :   T′ = ∅;  // T′ is a queue of test cases after prioritizing
2 :   S* = ∅;  // S* is a set of statements that has been
covered
3 :   for (each t ∈ T)
4 :     count the coverage Cover(t);
5 :   while (T′! = T )
6 :    {
7 :     if (S == ∅)
8 :     {
9 :      S = S*;
10 :     S* = ∅;
11 :    }
12 :    else
13 :    {
```

14 : t ∈ T and t covered the maximum subset S′of S;
15 : t join T' in the tail;
16 : $S = S - S'$;
17 : $S^* = S^* + S'$;
18 : }
19 : }

Then the prioritization value based on coverage is as follow:

$$C(t) = |Cover(t)| + \mu \quad (2)$$

When using additional strategy, all statements could be reset after being covered, and continue to calculate the coverage of the remaining test cases. Here may be a case that the coverage of test cases picked in the latter is higher than in the former. Therefore, in the actual calculation, each time after all statements being covered, make these test cases into one group and add the appropriate value μ based on the number of the total statements, making the entire group of test cases are given priority.

(2) Mutation analysis is used for FEP[11]. *S*, a set of statements, and $T = \{t_1, t_2, \cdots t_n\}$, a test suite. First, we create a set of mutants $N = \{n_1, n_2, \cdots n_m\}$ for $s \in S$. We use $\text{mutants}(s)$ as the total number of mutants of *S*. Then we execute each mutant version n_k of *N* on t_i, checking whether t_i kills that mutant. $\text{killed}(t_i)$ denotes as the number of mutants killed by t_i. Calculate the ratio of mutants killed as follows:

$$ms(s, t_i) = \frac{\text{killed}(t_i)}{\text{mutants}(s)} \quad (3)$$

So the prioritization value of t_i based on FEP is

$$FEP(t_i) = ms(s, t_i) \quad (4)$$

$FEP(t_i)$ is sorted by descending order and the larger the value, the higher the priority.

(3) For requirement property relevance, we calculate it as follows. Suppose that the requirement property of the system is a set $R = \{r_1, r_2, \cdots r_m\}$ and given a test suite $T = \{t_1, t_2, \cdots t_n\}$. For each requirement $r \in R$, there is at least one test case t in T which satisfies r. The satisfiability relation from T to R, denoted as $S(T, R)_{n \times m}$ and the algorithm of calculating *S* is as follows:

Algorithm 2 Calculate relationship S between test cases and requirement properties

1 : **for** (i=1; i<=n; i++)
2 : **for** (j=1; j <= m; j++)
3 : {
4 : **if** (t_i satisfies r_j)
5 : **then** $S(t_i, r_j) = 1$;
6 : **else** $S(t_i, r_j) = 0$;
7 : }

After the calculation, S can be considered as an n × m matrix. $\text{Req}(t_i)$ is used to denote the set of all requirements that satisfied by test case t_i, where $\text{Req}(t_i) = \{r | S(t_i, r) = 1\}$. Calculating $\text{Req}(t_i)$, you can know how many requirement properties each test case t_i satisfies, and $|\text{Req}(t_i)|$ is used to denote the number of Req .

The TCP calculation based on requirement property relevance is that to sort test cases according to$|\text{Req}(t_i)|$. So as follows:

$$RP(t_i) = |\text{Req}(t_i)| \quad (5)$$

(4) For the TCP calculation of historical information is somewhat complicated. We define H(t) as the history prioritization value, and the formula is as follows:

$$\begin{cases} H(t)_1 = 0 \\ H(t)_k = \alpha f_k + (1 - \alpha) H(t)_{k-1} \quad 0 \le \alpha \le 1 \ , \ k \ge 2 \end{cases} \quad (6)$$

Where $H(t)_k$ denotes the history prioritization value in the k-th regression testing. α is a smoothing constant used to weight individual history information, and here we take α = 0.8. f_k is the history information value in the k-th regression testing. For historical information value we can calculate it according to an objective (such as coverage). Can also do it according to the history of all the other objective information, as that make the last value PV_{k-1} as the history information f_k.Here, we choose the latter. The formula of f is

$$\begin{cases} f_1 = 0 \\ f_k = PV_{k-1} \quad (k > 1) \end{cases} \quad (7)$$

(5) The calculation based on time spending is relatively simple. Suppose that a test suite $T = \{t_1, t_2, \cdots t_n\}$, for each $t \in T$, calculate the execution time ,denoted as *tc*. The test cases are sorted according to the execution time. The calculation of *TC(t)* is as follows:

$$TC(t) = \frac{1}{tc} \quad (8)$$

Where, *tc* takes the reciprocal value in order to make *TC(t)* sorted according to the of the value. The greater the value, the higher the priority.

Due to the different measurement of each dimension, we need to process the results, which are calculated from each dimension, by normalization, in order to process the data conveniently.

Fig. 2. shows the computing process of *PV*.

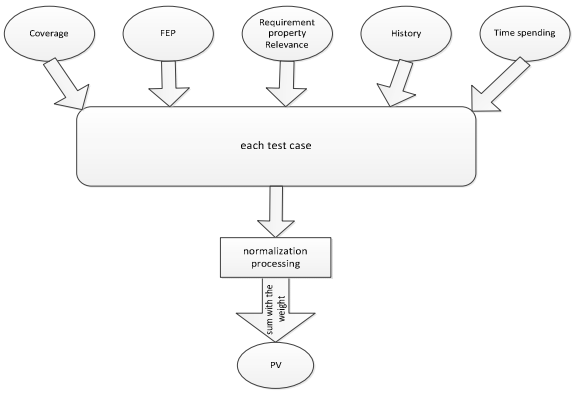

Fig. 2. computing process of PV

In Fig. 2, the top five ellipses denote objectives which based on the coverage, FEP, requirement property, historical information and time spending, respectively. The below arrows are the corresponding algorithms and the rounded rectangle denotes the test cases. Each objective calculates the test cases individually. The below rectangle is the normalization processing, and the last big arrow means the weighted sum of the above calculated values. *PV* is finally obtained in the below ellipse. The computation process is from multiple dimensions to a single dimension. For each test case, we calculate it on five dimensions, and then get the results from each objective. After the normalization, the values from every aspect are unified on one dimension to get the *PV* by the weighted sum.

C. Multi-objective Prioritization Algorithm

Suppose that a test suite $T = \{t_1, t_2, \cdots t_n\}$, each *t* in T is assigned a prioritization value, initially 0. Each prioritization is performed on all test cases to calculate values in each dimension so that to get *PV*. Here make the weight of every *PV* are equal. Then process the values by normalization. Finally, sort test cases by *PV*. The greater the *PV*, the higher the priority. Specific algorithm flow chart is shown in Fig. 3.

The algorithm first calculates the *PV* from the aspects from coverage, FEP, requirement property, historical information and time spending, respectively. Then normalize the results, the weighted sum of *PV* of five objectives is obtained. The weight is determined by actual requirement from system, the importance of each goal here is treated equally and the weight is taken as 0.2.

Each calculated *T'* will be stored in the database, which is convenient to query the archive and make the next prioritization.

IV. CASE STUDY

A. Calculation and Analysis

Given a procedure **P** and a test suite $T = \{t_1, t_2, t_3, t_4, t_5\}$, as follows.

Procedure P

```
1.  b = -b;
2.  while(a < 0) do
3.      if(b < 0)
4.        then exit;
5.      else b = b + 1;
            endif;
6.      a = a – 1;
        endwhile;
7.  if(a < 0)
          then
8.      { a = -a;
9.        b = b + 1;
          }
10. c = a + b;
```

test suite	
t1	a = -1, b = -1
t2	a = 1, b = 1
t3	a = 1, b = -1
t4	a = 0, b = 1
t5	a = 2, b = -2

What the execution path of each test case is can be known according to the test values. The statement coverage of them is as table 1.

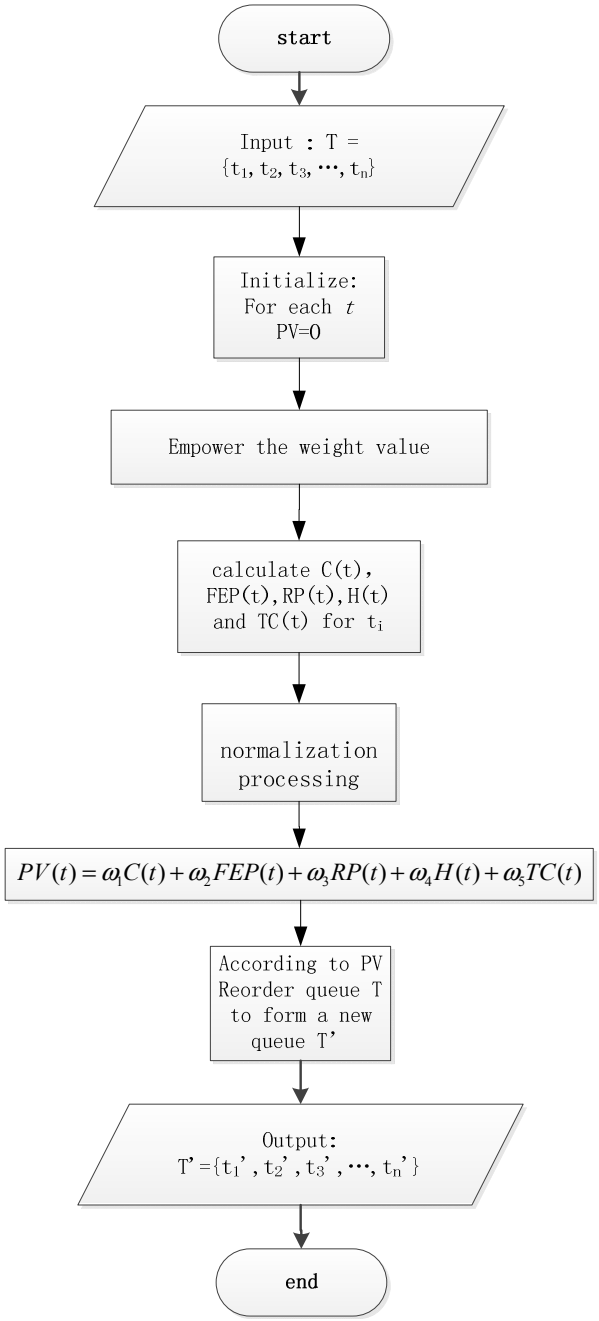

Fig. 3. TCP algorithm flow chart based on multi-objective

Table 1 statement coverage

	1	2	3	4	5	6	7	8	9	10
t1	x	x					x	x	x	x
t2	x	x	x	x						
t3	x	x	x		x	x	x			x
t4	x	x					x			x
t5	x	x	x		x	x	x			x

In the table 1, the sorting order based on additional strategy is (t3, t1, t2, t5, t4), and their coverage is 7,6,4,7 and 4,

respectively. Group ***a*** is picked out first, which contains t3, t2, t1. Group ***b*** has t5 and t4. Add 5 to all the members' C(t) of group ***a***, then we get C(t1) = 11, C(t2) = 9, C(t3) = 12, C(t4) = 4, C(t5) = 7.

The computation of FEP, requirement property and time spending is similar with the above. Because it is the first test in the regression testing, so there is no historical information, thus $H(t) = 0$. According to the equation defined in the previous chapters, the results of each dimension are shown in table 2.

Table 2 results of five dimensions

	C(t)	FEP(t)	RP(t)	H(t)	TC(t)
t1	11	4.43	6	0	1.7
t2	9	3.1	4	0	3.3
t3	12	4.25	7	0	1.4
t4	4	2.46	4	0	2.5
t5	7	4.25	7	0	0.9

Normalize the computation of each dimension, the result is shown in table 3.

Table 3 results of normalization processing

	C(t)	FEP(t)	RP(t)	H(t)	TC(t)
t1	0.256	0.24	0.214	0	0.173
t2	0.209	0.168	0.143	0	0.337
t3	0.279	0.23	0.25	0	0.143
t4	0.093	0.133	0.143	0	0.255
t5	0.163	0.23	0.25	0	0.092

Calculate the prioritization value of each test case with the five objectives, and figure 4 shows the results, t3 is better than others in coverage, but it cost more time than t1, t2 and t4.

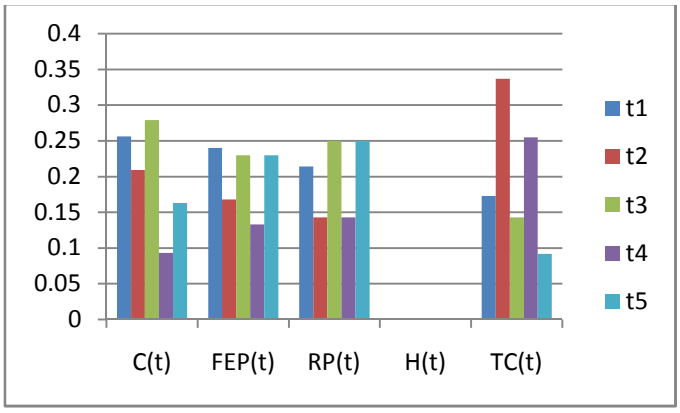

Fig. 4. comparison of results

As it's shown in Fig. 4, high coverage doesn't guarantee high faults detection rate, higher coverage cost more test time. The optimal efficiency can't be obtained by only one certain objective. Integrating multiple objectives is more practical and it meets the test requirements better.

Put the prioritization value of five dimensions into the formula (1),

PV(t1) = 0.2*0.256+0.2*0.24+0.2*0.214+0.2*0+0.2*0.173 = 0.1766

PV(t2) = 0.2*0.209+0.2*0.168+0.2*0.143+0.2*0+0.2*0.337 = 0.1714

PV(t3) = 0.2*0.279+0.2*0.23+0.2*0.25+0.2*0+0.2*0.143 = 0.1804

PV(t4) = 0.2*0.093+0.2*0.133+0.2*0.143+0.2*0+0.2*0.255 = 0.1248

PV(t5) = 0.2*0.163+0.2*0.23+0.2*0.25+0.2*0+0.2*0.092 = 0.1740

We sort the test cases by *PV* the order is (t3, t1, t2, t5, t4).

According to Fig. 4, if the test cases are sorted by coverage prioritization, the order is (t3, t1, t2, t5, t4); FEP prioritization's order is (t1, t3, t5, t2, t4); Requirement property relevance prioritization's order is (t3, t5, t1, t2, t4); Time spending prioritization's order is (t2, t4, t1, t3, t5).

B. *Effectiveness Measure*

The fitness metrics studied are based upon APFD (Average of the Percentage of Faults Detected)[2], which measures the weighted average of the percentage of faults detected over the life of the suite. However, it is not possible to know the faults exposed by a test case in advance. Therefore, we use APBC (Average Percentage Block Coverage)[12] in place of APFD. It is given by the equation

$$\text{APBC} = 1 - \frac{TB_1+TB_2+\cdots+TB_m}{nm} + \frac{1}{2n} \qquad (10)$$

Where n is the number of test cases and m is the number of statement blocks. TB_i is the first test case in the execution order T that covers statement block i.

We use APBC metrics to evaluate the case mentioned in the previous section. Fig. 5-10 shows the APBC values of unsorted, one single objective prioritization and multiple objective prioritizations. We use Fig. 5 as an example. After test case t1 executed, it covers 60% of the statement blocks and the corresponding point is (0.2, 60). Connecting all the points, the below area's ratio is the APBC.

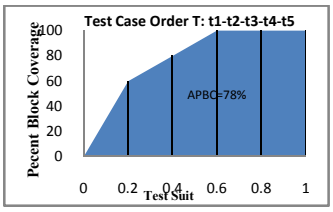

Fig. 5. APBC for no prioritization

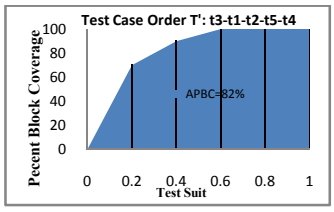

Fig. 6. APBC for coverage prioritization

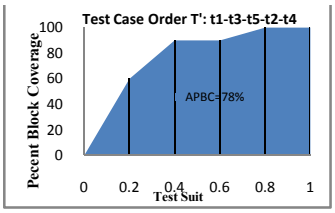

Fig. 7. APBC for FEP prioritization

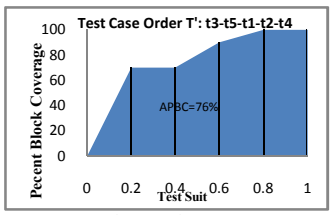

Fig. 8. APBC for requirement property relevance prioritization

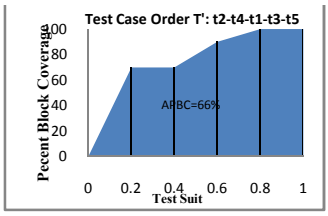

Fig. 9. APBC for time spending prioritization

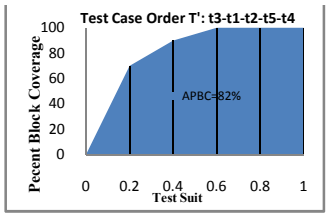

Fig. 10. APBC for multi-objective prioritization

The comparation result is shown in Fig. 11. Where plotted by the six sort methods on the horizontal axis and the APBC values on vertical, M1-M6 represents the TCP methods based on unsorted, coverage, FEP, requirement property relevance, time spending and multi-objective, respectively.

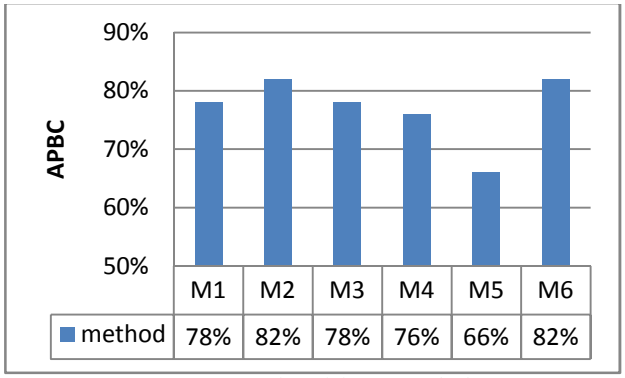

Fig. 11. comparation of six TCP methods

As Fig. 11 shows, both the proposed prioritization method and the method based on coverage have the maximum APBC, which demonstrates that our method is feasible and efficient. It also shows that if sorting them only by time spending, the performance is not the best. But if not considering time spending, the result could not be the best, sometimes would be the worst. So generally, time spending exists together with other objectives. Select the compromise is the best solution according to the golden mean. As the comparation of Fig. 11, we can see that the prioritization based on only one objective not always be the best performance, and the limitation of single objective sorting can be relieved by multi-objective prioritization. If further modest improvement and adjustment in each dimension, you can greatly optimize single objectives of defects in order to adapt to an increasingly complex regression testing of software system. This is also one of the purposes of this method proposed.

V. Conclusions and Future Work

The paper presents a dynamic test case prioritization method based on multi-objective. The proposed method first sorts the test cases focusing on five objectives on five dimensions, then calculate the weighted sum of their optimization results, finally sorts test suite according to the prioritization values. The proposed method is tested on a small experiment instance, and the results demonstrate that the optimized test case's APBC is better than the no prioritization. Due to the small scale of the experiment instance, it's unable to verify the effectiveness in large projects. The further work will be focused on larger scale program verification, and the weighted probability distribution and dynamic research can be considered, thus to improve this TCP method.

References

[1] Catal C, Mishra D. Test case prioritization: a systematic mapping study[J]. Software Quality Journal, 2013, 21(3): 445-478.

[2] Rothermel G, Untch R H, Chu C, et al. Prioritizing test cases for regression testing[J]. Software Engineering, IEEE Transactions on, 2001, 27(10): 929-948.

[3] Rothermel G, Untch R H, Chu C, et al. Test case prioritization: An empirical study[C]//Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on. IEEE, 1999: 179-188.

[4] Elbaum S, Malishevsky A G, Rothermel G. Prioritizing test cases for regression testing[M]. ACM, 2000.

[5] Elbaum S, Malishevsky A G, Rothermel G. Test case prioritization: A family of empirical studies[J]. Software Engineering, IEEE Transactions on, 2002, 28(2): 159-182.

[6] Srikanth H, Williams L. Requirements Based Test Case Prioritization[J]. IEEE Trans. on Software Engineering, 2002, 28.

[7] Srikanth H, Williams L, Osborne J. System test case prioritization of new and regression test cases[C]//Empirical Software Engineering, 2005. 2005 International Symposium on. IEEE, 2005: 10 pp.

[8] Kim J M, Porter A. A history-based test prioritization technique for regression testing in resource constrained environments[C]//Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on. IEEE, 2002: 119-129.

[9] Zhang X, Nie C, Xu B, et al. Test case prioritization based on varying testing requirement priorities and test case costs[C]//Quality Software, 2007. QSIC'07. Seventh International Conference on. IEEE, 2007: 15-24.

[10] Huang Y C, Peng K L, Huang C Y. A history-based cost-cognizant test case prioritization technique in regression testing[J]. Journal of Systems and Software, 2012, 85(3): 626-637.

[11] Chen X, Chen JH, Ju XL, Gu Q. Survey of test case prioritization techniques for regression testing. Ruan Jian Xue Bao/Journal of Software, 2013,24(8):1695−1712 (in Chinese).
http://www.jos.org.cn/1000-9825/4420.htm

[12] Li Z, Harman M, Hierons R M. Search algorithms for regression test case prioritization[J]. Software Engineering, IEEE Transactions on, 2007, 33(4): 225-237.

[13] Fraser G, Wotawa F. Test-case prioritization with model-checkers[C]//25th conference on IASTED International. 2007.

[14] Harman M. Making the Case for MORTO: Multi Objective Regression Test Optimization[C]//ICST Workshops. 2011: 111-114.

[15] Yoo S, Harman M. Regression testing minimization, selection and prioritization: a survey[J]. Software Testing, Verification and Reliability, 2012, 22(2): 67-120.