MFTCP: Multiple Factors based Test Case Prioritization

Can Cui[1][0000-0003-3199-2834], Wang Zhi[1], Wang Shihai[2*][0000-0002-0241-0009] and Liu Bin[2]

[1] North China Institute of Computing Technology, Beijing 100083, China
[2] Beihang University, Beijing 100191, China
*wangshihai@buaa.edu.cn

Abstract. In software regression testing, reorder the test cases to ensure the modified code does not introduce new effects and keep function right. Traditional coverage-based test case prioritization (TCP) approaches are Total and Additional. These two approaches only use code coverage as the measure to rank the test cases. Several defect prediction-based TCP approaches integrate the defect prediction probability and code coverage with weighting coefficients. However, previous methods need to have a priori knowledge related to defects, which is difficult to obtain. Therefore, we proposed an approach named MFTCP by consider code coverage, defect prediction probability, and lines of code (LoC) with automatic weighting coefficients to rank regression test cases. The experimental results indicate the proposed method is more effective than the baselines, especially in terms of the total strategy.

Keywords: Software Testing, Test Case Prioritization (TCP), Code Coverage, Defect Prediction (DP).

1 Introduction

When the software initial test (SIT) has been completed, the execution information of test cases is known, i.e., the coverage and the execution results of the test cases are obtained. Software regression testing (SRT) is used to confirm that recent program changes have not adversely affected existing functionality, and new tests must be performed to test the new functionality [1]. In white-box SRT, the popularly traditional dynamic code coverage-based test case prioritization (TCP) techniques are Total and Additional. However, the techniques only utilize the dynamic code coverage information without the execution results.

At present, there are several studies that use the outputs of test cases and/or static metrics of software program to integrate with dynamic code coverage to prioritize test cases. For example, the Modified code coverage method [2] (Modified), Quality-aware TEst case Prioritization (QTEP) [3], Adaptive TCP method (AdaTCP) [4], etc. These methods use static metrics to build defect prediction models (DPMs) to achieve the defect prediction probability (DPP) and conduct a variant object function to integrate code coverage and prediction probability. However, these methods require the prior knowledge of software defects (such as the number of defects in the software). Besides, the setting of the coefficient of multiple factors (such as Modified 2]) is difficult. The coefficient of the software is difficult to obtain and usually fixed. Different software program has different defect distribution, which makes the fixed coefficient improper.

Therefore, multiple factors-based test case prioritization method (MFTCP) is proposed. MFTCP considers DPP, software static metrics, and dynamic code coverage. In this method, the three factors of are weighted to conduct a novel objective function. The weighted coefficient is obtained by the performance evaluation measure of DPM, rather than by the prior knowledge setting. In the paper, Balance is selected as one of the coefficients. Because DPMs are varied according to different software under test (SUT). Therefore, the Balance value changes with the change of DPM.

2 Related Works

2.1 Traditional Test Case Prioritization Approaches

Code coverage-based TCP was proposed in the early 21st century. From 1999 to 2002, Rothermel and Elbaum published several papers to form the main framework of code coverage-based TCPs [5][6][7]. Total and Additional strategies implemented by the greedy algorithm are the most traditional approaches [5]. Coverage-based TCP methods use the coverage information of source code and test cases (i.e., static code coverage information or dynamic code coverage information), and rank test cases by maximizing code coverage by adopting different coverage criteria [8]. The dynamic code coverage is collected by executing the program and tracking each executed unit [9]. Commonly used coverage criteria are statement coverage [5], function/method coverage, branch coverage [5], block coverage, data-flow coverage, and Modified Condition/Decision Condition (MC/DC) coverage etc.

Zhang et al. [10][11] used the probability model to merge Total and Additional into a unified model, and proposed the Basic model and Extended model to bridge the gap between Total and Additional. The basic idea is that "each time a unit is covered by a test case, which can potentially detect some faults in the unit, the probability that the unit contains undetected faults decreases by certain rate". Experiments show that using uniform probability values with multiple strategies can significantly outperform Total and Additional.

2.2 Hybrid Test Case Prioritization Approaches

Total and Additional only consider code coverage, that is, the assumption is that "test cases with higher coverage can detect more defects". The assumption implies that "defects are evenly distributed in the program", which is obviously inconsistent with the 20:80 Rule [12][13]. Based on this, several studies [5][6][14][15] have proposed to prioritize test cases by considering multiple factors including code coverage.

Elbaum et al. [6] considered test cost and fault severity level, and used three fault severity levels and five test cost distributions for case application. Among them, the linear method sets six fault severity levels, that is, the fault level value is from 1 to 6, and the fault level value is set from 20 to25 using the exponential method.

Rothermel et al. [5] consider the ability of a test case to expose faults depending not only on whether the test case reaches the faulty statement, but also on the probability that the fault in the statement causes the test case to fail. This paper proposes Total Fault-Exposing Potential coverage (FEP-total) and Additional Fault-Exposing Potential coverage (FEP-additional). Approximate FEP values of test cases are obtained using mutation analysis.

Recent studies have shown that even the optimal coverage-based TCP method does not outperform the Additional method much in terms of fault detection rate [16]. Hao et al. [16] argue that to improve the Additional method, some information other than the structure coverage should be adopted. Paterson et al. [17] believe that part of the reason may be the complexity of real faults and the assumption of coverage-based methods. Therefore, some studies have proposed some improved methods by fusing static metrics and test case related information [3][14].

Wang et al. [3] proposed QETP that integrates code inspection method to solve the problem that the coverage-based TCP methods only focus on maximizing coverage and lacks attention to the distribution of defects in the source code. They linearly ensemble the static bug finder FindBugs [18] and the static code coverage-based method JUPTA [9], and adopted the cross-project defect prediction with common metrics (CPDP-CM) model CLAMI [19] with the dynamic code coverage-based methods (Total and Additional). The experimental results show that QTEP can improve the defect detection rate on both regression test cases and new test cases.

Mahdieh et al. [2] proposed Modified by combining cross version defect prediction (CVDP) model based on self-defined neural network and coverage-based TCP method. Modified uses the modified coverage criteria for the Total and Additional strategies, respectively. This method assumes that the fault probability of the software unit is known, that is, in the unit set of the source code $U=\{u_1,u_1,\cdots,u_m\}$, the j_{th} code unit u_j ($1 \le j \le m$) contains faults and F_j represents the event where the fault occurs, and the probability of the event failure $P(F_j)$ is known. Then, the modified code coverage calculation method of the test case t_i is as shown in the formula (1), that is, the test case covering the fault unit is added with a greater weight value, so that its priority becomes higher.

$$\begin{cases} FaultBasedCover(i) = \sum_{1\le j\le m} Cover(i,j)\times P(F_j) \\ P(F_j) = P_0 + (1\text{-}P_0)\times P_{DP}(j) \\ P_0 = 1\text{-}C_{DP} \end{cases} \tag{1}$$

Where, C_{DP} represents prior knowledge (e.g., the number of bugs in history), $P_0 \in [0,1]$. Let $P_{DP}(j)$ denote the defect probability of the unit u_j predicted using the DPM; When the project provides more prior knowledge, C_{DP} can be set to a higher value to increase the influence of prior knowledge. Otherwise, the value of C_{DP} should be decreased to reduce the influence of prior knowledge.

Existing defect prediction-based TCP methods consider both defect distribution and code coverage, which breaks the implicit assumption that defects in software are uniformly distributed. However, these methods only consider defect distribution and code coverage, besides, the weighting coefficient is fixed according to the prior knowledge related to the defect, which is difficult to obtain. Moreover, different prior knowledge makes test case ranking difference, which leads to the narrow application scope of this kind of method. Therefore, it is worth studying how to integrate multiple factors such as defect distribution information and code coverage for TCP to have wider applicability and better performance.

3 The proposed Approach

3.1 The Overview of The Approach

The framework of the proposed MFTCP is shown as Fig. 1. From the Fig. 1, we can see that the approach contains three stages:

Firstly, **build defect prediction model.** The historical static metric information and the output results of test cases are used to construct the DPM, the DPP is obtained and the credibility of the DPM is evaluated. Secondly, **construct objective function.** The defect probability, the most popular static metric (LoC) and the code coverage are integrated by the weighted method to obtain the objective function of TCP. Finally, **prioritize test cases**. The greedy algorithm combined with the objective function is used to get the prioritized test cases, and the performance evaluation of TCP is obtained.

3.2 Defect Prediction Model

We use machine learning methods to build DPMs. The test data is the current version of the system under test (SUT). The training data is historical or cross-project versions, and the validation data is the last version of the current version. According to the data, the performance value of the DPM is obtained by validation data. Besides, the defect probability of the SUT is also obtained.

3.3 Objective Function Construction

Multiple Factors Selection. Three factors (i.e., dynamic code coverage, LoC and defect distribution) are adopted to construct objective function to generate the test case rank. The defect distribution is obtained by the DPM, that is, the defect probability of the software entity is obtained. The granularity of entities is determined by the granularity of software metrics. For example, the granularity in object-oriented software is usually at the class level, and the granularity in process-oriented software is usually at the method level. The software entity the defect probability belongs to coincides with the one measured by LoC. LoC are determined by the granularity of software entities, i.e., if the entity granularity in the DPM is the class level, then the line of code refers to the line of class code. If the entity granularity is the method level, then the LoC refers to the line of method code. Dynamic code coverage is the count of software entities (such as statement, branch and method) covered by test cases.

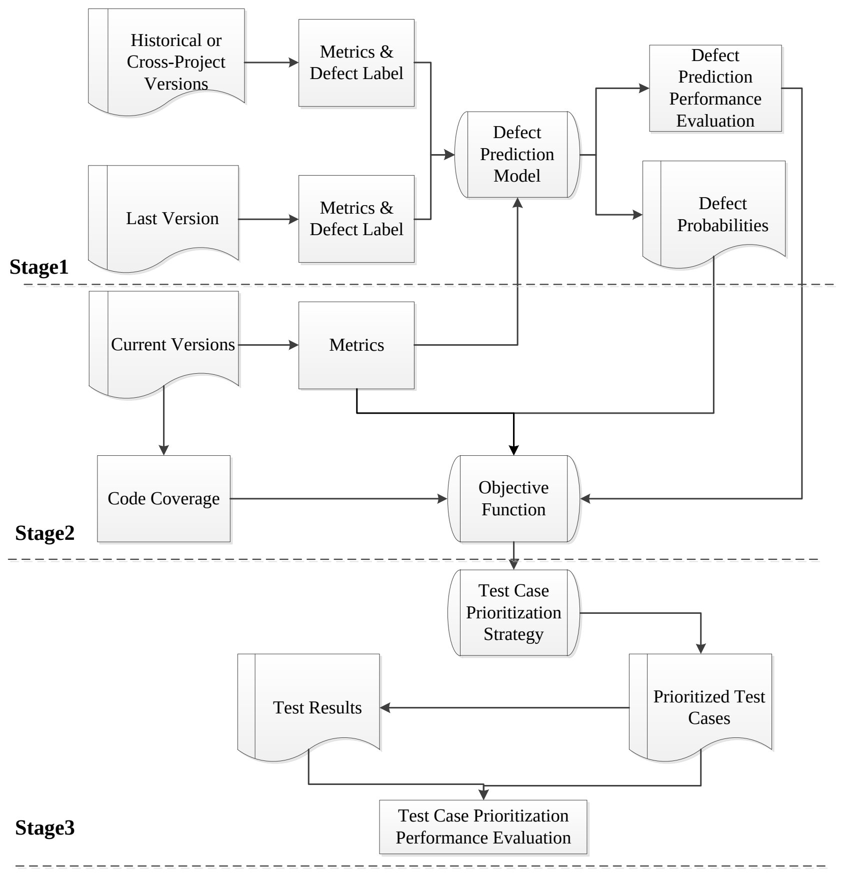

Fig. 1. The framework of the proposed MFTCP.

Linearly Integrate Multiple Factors. To avoid the influence between defect distribution and dynamic code coverage, linear function is obtained. Since the granularity of defect distribution is consistent with LoC, the number of lines of code is linearly normalized and multiplied with the defect probability to as an integrated factor. Besides, to avoid the "cancellation" effect after normalization to 0, the number of LoC is normalized to [0.5, 1] to obtain the objective function for test cases, as shown in the formula (2).

$$\begin{cases} f = w_1 * (P_{dp} * m) + w_2 * C_{tc} \\ m = 1 - 0.5 * \dfrac{(metrics - metrics_{\min})}{metrics_{\max} - metrics_{\min}} \end{cases} \tag{2}$$

where, $metrics$ represents the value of static metric (i.e., LoC), m represents the normalized value ranging from [0.5, 1], P_{dp} means the defect probability, C_{tc} stands for code coverage of program, and w_1 (w_2) represents the weighting coefficient of defect probability times multiple LoC (code coverage).

Weighting Coefficients Determination. To avoid setting the weighting coefficients is relied upon priori knowledge like MAP [2] and Basic [10][11], we use the validation set to evaluate performance of defect models. Besides, Balance proposed by Menzies et al [20] is as the weighting coefficient for the probability of defect prediction and LoC. The reasons for choosing to the performance measure are: (a) the defect distribution of the data is very close or even the same in the same project, although the version has been changed. Besides, CVDP can well predict for the defect distribution in the next version (new version) in the same project, and get the defect probability of the software entity; (b) the performance measure varies according to the validation set (the previous version of the test set), that is, for different target projects, the credibility of the constructed DPM is not always 100% or is not a constant value. Therefore, the choice of performance measure is consistent with the need for a multi-factor linearly weighted integration that does not depend on a priori knowledge. (c) ***Balance*** is for the assessment of the performance of DPMs for classification under the unbalanced data condition. Besides, it is a composite metric used in many DPMs. Moreover, the coefficient does not have the null value and takes values in the range [0, 1].

3.4 Test Case Prioritization Strategy

Based on the objective function, the Total and Additional strategies are used. According to the function and methods, the rank of test cases is achieved. In general, the granularity of dynamic code coverage does not coincide with the granularity of software entities in DPMs. Therefore, the entity granularity covered in the test case and the entity granularity of the defect prediction metric need to be converted to the same dimension, and then calculated the integrated values to reorder test cases. After SRT is finished, the outputs of the test cases are obtained. The defect detection rate of the TCP method can be calculated. That is, the performance measure of MFTCP can be gotten.

4 The Experimental Study

4.1 Research Questions

To verify the performance of the proposed method, the following two research problems are set up:

RQ1: Does the proposed MFTCP perform better than the other defect prediction-based TCP methods?

RQ2: Does the proposed MFTCP perform better than the traditional coverage-based TCP methods?

To verify the effectiveness of the proposed method, from the perspective of defect detection rate, research questions 1 and 2 were set. In RQ1, MFTCP is compared with Modified[2] in CVDP scenario and QTEP[3] in CPDP-CM scenario. In RQ2, MFTCP is compared with the Total, Additional, Original Order Prioritization (ORP), Reverse Order Prioritization (REP), RAndom Prioritization (RAP) and OPtimal Prioritization (OPP).

4.2 Datasets

A total of 165 versions of five Java projects are used, which are in the public Defect4J+M [2]. Two different scenarios named CVDP and CPDP-CM are considered, the benchmark data are shown in Table 1.

Table 1. Dataset usage for the two scenarios CVDP and CPDP-CM

Scenario	Project	Version Number for Test Data	Version Number for Training Data	Version Number for Validation Data
CVDP	Chart	1-15	3-26	2-16
	Closure	1-15	3-133	2-16
	Lang	1-15	3-65	2-16
	Math	1-15	3-106	2-16
	Time	1-15	3-27	2-16
CPDP-CM	Chart	1-15	Closure/Lang/ Math/Time 1-15	Chart 2-16
	Closure	1-15	Chart /Lang/ Math/Time 1-15	Closure 2-16
	Lang	1-15	Chart/Closure / Math/Time 1-15	Lang 2-16
	Math	1-15	Chart/Closure / Lang/Time 1-15	Math 2-16
	Time	1-15	Chart/ Closure/ Lang/Math 1-15	Time 2-16

4.3 Evaluation Measures

First-Failing Rate (FFR): It is the ratio of the number of test cases needed to find the first defect in the software to the total number of test cases. As shown in the formula (3), the smaller the FFR, the better the performance of TCP.

$$FFR = \frac{|test_{ff}|}{|test_v|} \tag{3}$$

Where $|test_{ff}|$ is the number of test cases used to discover the first defect in the current version of the software and $|test_v|$ is the total number of test cases in the current version.

Weighted Average of the Percentage of Faults Detected (APFD): It is a popular measure used to calculate the defect detection rate of TCP. The formula is shown in Equation (4), which ranges from 0 to 1.

$$\begin{aligned} APFD &= 1 - \frac{TF_1 + TF_2 + \cdots + TF_m}{mn} + \frac{1}{2n} \\ &= 1 - \frac{\sum_{i=1}^{m} TF_i}{mn} + \frac{1}{2n} \end{aligned} \tag{4}$$

Where TF_i represents the location index of the ranked test cases that finds the first defect i . The higher the APFD value, the better the TCP approach.

In this paper, the performance of TCP is evaluated using FFR, APFD.

4.4 Parameter Settings

To maintain fairness, the parameters settings are as the description of the method in the original studies. For example, in the Modified method [2], the parameters in the custom neural network are consistent with the original paper, and under-sampling (defective: non-defective =1:2) is used to process the data for class imbalance. The weighted coefficient values in Modified[2] and QETP[3]are all the best values in the original paper, or the values in the best range.

5 Results and Discussion

5.1 RQ1

RQ1.1 Does MFTCP outperform Modified? In the CVDP scenario, MFTCP and Modified use the Additional and Total strategies to prioritize the test cases in a total of 75 versions of five projects (i.e., Chart, Closure, Lang, Math and Time). The performance evaluation results of the methods are obtained.

The results of APFD and FFR obtained of MFTCP and Modified with Additional and Total strategies are shown in Table 2 and Table 3.

As shown in Table 2 and Table 3, when using the Total strategy, MFTCP obtains better values of APFD and FFR than the comparison method Modified on each project for 5 projects. Besides, with respect to APFD and FFR, MFTCP is superior to Modified on each of five projects (APFD: 58.53%>57.74%; FFR: 39.4%<40.46%). For the Additional strategy, in terms of APFD and FFR, MFTCP performs better than Modified on most of five projects. Moreover, MFTCP with Additional performs better than Modified with Additional on average (APFD: 57.63%>57.17%; FFR: 37.5%<38.79%).

Table 2. The average APFD of MFTCP and Modified.

Project	MFTCP	Modified	MFTCP	Modified
	Total (%)		Additional (%)	
Chart	**61.31**	61.11	58.03	**58.83**
Closure	**66.92**	66.47	**78.04**	76.30
Lang	**49.71**	49.24	38.63	**38.82**
Math	**56.73**	55.93	57.29	**58.49**
Time	**57.98**	55.94	**56.15**	53.43
Avg.	**58.53**	57.74	**57.63**	57.17

Note: The bold data represents the better method between MFTCP and Modified.

Table 3. The average FFR of MFTCP and Modified.

Project	MFTCP	Modified	MFTCP	Modified
	Total (%)		Additional (%)	
Chart	**37.0**	37.27	39.4	**37.81**
Closure	**30.5**	31.14	**16.6**	20.22
Lang	**49.7**	50.13	56.4	**56.18**
Math	**40.4**	42.68	**38.9**	39.01
Time	**39.1**	41.07	**36.0**	40.71
Avg.	**39.4**	40.46	**37.5**	38.79

Note: The bold data represents the better method between MFTCP and Modified.

From the above analysis of the results in the CVDP scenario, it is can be known that MFTCP outperform the comparison method Modified on average of most projects, whether combined with the Additional or Total strategy. The results indicate that combining LoC and defect probability with code coverage can improve the defect detection rate, especially in terms of the Total strategy.

RQ1.2: Does MFTCP outperform QTEP? In the CPDP-CM scenario, MFTCP and QTEP use the Additional and Total strategies on a total of 75 versions of 5 projects and obtain the corresponding performance evaluation results. The results of APFD and FFR are shown in Table 4 and Table 5.

In terms of APFD which is shown in Table 4, in terms of APFD, MFTCP with Total are better than QTEP with Total on average of five projects (57.02>56.39). Besides, and the values of APFD for MFTCP are larger than the values for QTEP on most of the five projects. However, MFTCP with Additional performs better than QTEP with Additional on Closure and Math, but MFTCP with Additional is worse than QTEP with Additional on average and the other three projects.

Table 4. The average APFD of MFTCP and QTEP.

Project	MFTCP	QTEP	MFTCP	QTEP
	Total (%)		**Additional (%)**	
Chart	**59.94**	58.61	57.60	**60.52**
Closure	**66.57**	66.37	**77.97**	77.80
Lang	**47.12**	45.60	37.67	**42.75**
Math	56.38	56.42	**59.37**	52.75
Time	**55.08**	54.92	53.80	**72.22**
Avg.	**57.02**	56.39	57.28	**61.21**

Note: The bold data represents the better method between MFTCP and QTEP.

Table 5. The average FFR of MFTCP and QTEP.

Project	MFTCP	QTEP	MFTCP	QTEP
	Total (%)		**Additional (%)**	
Chart	**38.6**	40.31	39.5	**35.83**
Closure	**30.8**	31.10	**16.6**	19.28
Lang	52.2	**53.82**	58.9	**54.04**
Math	41.9	**41.50**	**36.9**	44.64
Time	**42.5**	42.77	37.4	**19.44**
Avg.	**41.2**	41.90	37.8	**34.64**

Note: The bold data represents the better method between MFTCP and QTEP.

With respect to FFR as shown in Table 5, for the Total strategy, MFTCP has better FFR than QTEP on most projects. Besides, the average values of FFR for MFTCP and QTEP are 41.2% and 41.90%, respectively. For the Additional strategy, MFTCP only outperforms QTEP on Closure (16.6%<19.28%) and Math (36.9%<44.64%). MFTCP with Additional has worse performance on Chart, Lang and Time in terms of APFD and FFR. The main reason is probably that the additional defect probability is biased, which decreases the defect rate for test cases.

From the above analysis of the results in the CPDP-CM scenario, In terms of APFD and FFR, MFTCP combined with Total outperforms QTEP, while MFTCP using the Additional strategy outperforms QTEP on Closure and Math according to the values of APFD and FFR.

In summary of RQ1.1 and RQ1.2, in the CVDP scenario, in terms of Total and Additional, MFTCP combined with the Total strategy outperforms the baselines. In the CPDP-CM scenario, with respect to the Total strategy, MFTCP performs better than the compared approaches. Besides, in terms of Additional, MFTCP performs better than the baseline methods on part of the datasets.

5.2 RQ2

RQ2.1 Does MFTCP combined with Total outperform the traditional baselines? For MFTCP, the CVDP and CPDP-CM scenarios are considered. The APFD values of MFTCP in the two scenarios and the baseline approaches on five projects and on average are listed in Table 6.

Table 6. The average APFD (%) of MFTCP with Total and the baselines.

Project	MFTCP$_{CV}$	MFTCP$_{CP}$	Total	ORP	REP	RAP	OPP
Chart	61.31	59.94	**61.68**	46.68	53.32	52.35	99.93
Closure	**66.92**	66.57	66.65	51.89	48.11	50.14	99.99
Lang	49.71	47.12	48.87	44.51	**55.49**	52.01	99.97
Math	56.73	56.38	57.00	**58.30**	41.70	51.28	99.96
Time	**57.98**	55.08	55.37	45.65	54.35	49.36	99.97
Avg.	**58.53**	57.02	57.91	49.41	50.59	51.03	99.97

Note: The bold data represents the best method except OPP.

As shown in Table 6, the OPP is a perfect method, which reaches the best value of APFD. All values are above 99.90%. ORP and REP are two special cases of RAP. From the results, we can know that the APFD values of ORP, REP and RAP are all close to 50%, which is lower than MFTCP in the CVDP scenario (58.53%) and the CPDP-CM scenario (57.02%). In the CVDP scenario, the average value of APFD on five projects of MFTCP is 58.53%, which is the second-best value among these methods. That is, MFTCP using the CVDP model performs the best among Total (57.91%), Original (49.41%), Reverse (50.59%) and Random (51.03%). Besides, MFTCP has two second best values of APFD on Closure (66.92%) and Tine (57.98%). MFTCP does not outperform Total on Chart and Math. In the CPDP-CM scenario, MFTCP performs close to Total, which has the third best value on average (i.e., 57.91% of Total). Moreover, MFTCP is close to Total on most projects, such as Closure and Time. From the results, we can see that MFTCP of CVDP perform better than Total but MFTCP of CPDP-CM performs worse than Total on Closure, Lang and Time. The reason is probably that the defect prediction results of CVDP are better than the results of CPDP-DM.

RQ2.2 Does MFTCP combined with Additional outperform the traditional baselines? Similarly to RQ2.1, the APFD results for MFTCP with Additional by considering the CVDP and the CPDP-CM scenarios and the five baseline methods are shown in Table 7.

As shown in Table 7, the values of ORP, REP, RAP and OPP are the same as in Table 6. The values of APFD for MFTCP in CVDP and CPDP-CM scenarios are 57.63% and 57.28%, respectively. On average, MFTCP are better than ORP, REP and RAP. In the CVDP scenario, MFTCP performs better than Additional on Chart (58.03%>56.63%) and Math (57.29%>49.57%), but worse than Additional on Closure (78.04%<84.95%), Lang (38.63%<42.91%) and Time (56.15%<59.57%). Similarly, in the CPDP-CM scenario, MFTCP outperforms Additional on Chart and Math, but does not outperform Additional on Closure, Lang and Time. The defect prediction probabilities make an effect on the proposed TCP method. Therefore, the reason may be that the biased defect prediction results with the additional strategy decreases the defect detection rate on some versions.

Table 7. The average APFD (%) of MFTCP with Additional and the baselines.

Project	MFTCP_{CV}	MFTCP_{CP}	Additional	ORP	REP	RAP	OPP
Chart	**58.03**	57.60	56.63	46.68	53.32	52.35	99.93
Closure	78.04	77.97	**84.95**	51.89	48.11	50.14	99.99
Lang	38.63	37.67	42.91	44.51	**55.49**	52.01	99.97
Math	57.29	**59.37**	49.78	58.30	41.70	51.28	99.96
Time	56.15	53.80	**59.57**	45.65	54.35	49.36	99.97
Avg.	57.63	57.28	**58.77**	49.41	50.59	51.03	99.97

Note: The bold data represents the best method except OPP.

In summary of RQ2.1 and RQ2.2, in the CVDP scenario, MFTCP combined with the Total strategy outperforms the baselines (Total, ORP, REP and RAP) except OPP. In the CPDP-CM scenario, MFTCP with Additional can achieve better APFD than pure Additional on some projects.

5.3 Discussion

The method combines defect prediction results, a metric of software program (i.e., LoC) and dynamic code coverage to form an ensemble approach. The prediction result is a key factor of MFTCP. To reach the result, the prepare work is needed, such as metrics collection, datasets construction and processing, defect prediction model construction. In the experiment, we use five projects with different sizes and types. For example, Math has at least (most) 905 (1105) instances for defect prediction and 3458 (4270) test cases with 4384 (5252) units for prioritization. Lang is with at least (most) 214 (230) instances for defect prediction and 2049 (2291) test cases and 2018 (2197) units for prioritization. For the experimental results, the proposed MFTCP indicates the effectiveness, which the numbers instances from the projects are larger than 200. For smaller projects, such as the project with below 100 instances, constructing machine learning model is difficult with few samples. Moreover, preparing datasets is time-consuming. Therefore, MFTCP might be improper for the smaller projects.

6 Conclusion and Future Work

In the SRT phase, defect prediction-based TCP methods integrate the defect prediction probability and code coverage by the weighting factors. But the weighting values need to have a priori knowledge that is difficult to obtain, such as those related to defects, etc. Therefore, a multi-factorial linear weighting test case prioritization method named MFTCP is proposed. MFTCP takes the defect distribution (i.e., defects), LoC, and the code coverage of test cases into account. Besides, an objective function is constructed by integrating the defect prediction probability and LoC with the code coverage in a linearly weighted way. The former coefficient is calculated by using the DPM evaluation measure ***Balance***, and the latter as 1-*Balance*. Since ***Balance*** varies according to different target project, a three-factor linearly weighted integration can be achieved instead of setting it based on a priori knowledge.

To verified the effectiveness of MFTCP, both CVDP and CPDP-CM scenarios are considered for construct DPMs and extensive experiments on a total of 75 versions of five publicly available Java projects (Chart, Closure, Lang, Math and Time) in Defect4J+M. The experimental results demonstrate the effectiveness of MFTCP combined with two traditional strategies, Total and Additional. Especially, MFTCP outperforms the compared method Modified in terms of APFD and FFR.

Acknowledgments. No funding was received to assist with the study.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

- References

1. Wong, W. E., Horgan, J. R., London, S., et al.: A Study of Effective Regression Testing in Practice. In:8th International Symposium on Software Reliability Engineering (ISSRE1997) on Proceedings, pp. 264-274. IEEE Computer Society, Los Alamitos (1997)
2. Mahdieh M., Mirian-Hosseinabadi S.-H., Etemadi K., et al.: Incorporating Fault-Proneness Estimations into Coverage-based Test Case Prioritization Methods. Information and Software Technology **121**, 106269 (2020)
3. Wang S., Nam J., Tan L.: QTEP: Quality-Aware Test Case Prioritization. In: 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2017) on Proceedings, pp. 523-534. Association for Computing Machinery, New York (2017)
4. Hao D., Zhao X., Zhang L.: Adaptive Test-Case Prioritization Guided by Output Inspection. In: 37th Annual Computer Software and Applications Conference (COMPSAC 2013), pp. 169-179. IEEE Computer Society, Los Alamitos (2013)
5 Rothermel G., Roland H. U., Chu C., et al.: Prioritizing Test Cases for Regression Testing. IEEE Transactions on Software Engineering **27**(10), 929-948 (2001)
6 Elbaum S., Malishevsky A.G., Rothermel G.: Test Case Prioritization: A Family of Empirical Studies. IEEE Transactions on Software Engineering, **28**(2), 159-182 (2002)

7 Rothermel G., Roland H U., Chu C., et al.: Test Case Prioritization: An Empirical Study. In: IEEE International Conference on Software Maintenance (ICSM'99) on Proceedings, pp. 179-188. Institute of Electrical and Electronics Engineers Inc., United States (1999)
8 Luo Q., Moran K., Poshyvanyk D.: A Large-Scale Empirical Comparison of Static and Dynamic Test Case Prioritization Techniques. In: 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016) on Proceedings, pp. 559-570. Association for Computing Machinery, New York (2016)
9 Mei H., Hao D., Zhang L., et al.: A Static Approach to Prioritizing JUnit Test Cases. IEEE Transactions on Software Engineering **38**(6), 1258-1275 (2012)
10 Zhang L., Hao D., Zhang L., et al.: Bridging the Gap between the Total and Additional Test-Case Prioritization Strategies. In: 35th International Conference on Software Engineering (ICSE2013) on Proceeding, pp. 192-201. Institute of Electrical and Electronics Engineers Inc., United States (2013)
11 Hao D., Zhang L., Zhang L., et al.: A Unified Test Case Prioritization Approach. ACM Transactions on Software Engineering and Methodology **24**(2), 1-31 (2014)
12 Boehm B., Basili V.R.: Software Defect Reduction Top 10 List. Computer **34**(1), 135-137 (2001)
13 Shull F., Basili V., Boehm B., et al.: What We Have Learned About Fighting Defects. In: 8th IEEE Symposium on Software Metrics (METRICS 2002) on Proceedings, pp. 249-258. IEEE Computer Society, Los Alamitos (2002)
14 Eghbali S., Kudva V., Rothermel G., et al.: Supervised Tie Breaking in Test Case Prioritization. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion (ICSE-Companion 2019) on Proceedings, pp. 242-243. Institute of Electrical and Electronics Engineers Inc., United States (2019)
15 Paterson D., Campos J., Abreu R., et al.: An Empirical Study on the Use of Defect Prediction for Test Case Prioritization. In: 2019 IEEE 12th Conference on Software Testing, Validation and Verification (ICST2019) on Proceedings, pp. 346-357. Institute of Electrical and Electronics Engineers Inc., United States (2019)
16 Hao D., Zhang L., Zang L., et al.: To Be Optimal or Not in Test-Case Prioritization. IEEE Transactions on Software Engineering **42**(5), 490-505 (2016)
17 Paterson D., Kapfhammer G.M., Fraser G., et al.: Using Controlled Numbers of Real Faults and Mutants to Empirically Evaluate Coverage-Based Test Case Prioritization. In: 13th International Workshop on Automation of Software Test (AST 2018) on Proceedings, pp. 57-63. Institute of Electrical and Electronics Engineers Inc., United States (2018)
18 Hovemeyer D., Pugh W.: Finding Bugs Is Easy. ACM Sigplan Notices **39**(12), 92-106 (2004)
19 Nam J., Kim S.: CLAMI: Defect Prediction on Unlabeled Datasets. In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE2015) on Proceedings, pp. 452-463. Institute of Electrical and Electronics Engineers Inc., United States (2015)
20 Menzies T., Greenwald J., Frank A.: Data Mining Static Code Attributes to Learn Defect Predictors. IEEE Transactions on Software Engineering **33**(1), 2-13 (2007)