A Clustering – Bayesian Network Based Approach for Test Case Prioritization

Xiaobin Zhao, Zan Wang*, Xiangyu Fan
School of Computer Software
Tianjin University
Tianjin, China
{zhaoxiaobin, wangzan, fxy}@tju.edu.cn

Zhenhua Wang
American Electric Power
Gahanna, OH, USA
zhw.powersystem@gmail.com

***Abstract*—Test case prioritization can effectively reduce the cost of regression testing by executing test cases with respect to their contributions to testing goals. Previous research has proved that the Bayesian Networks based technique which uses source code change information, software quality metrics and test coverage data has better performance than those methods merely depending on only one of the items above. Although the former Bayesian Networks based Test Case Prioritization (BNTCP) focusing on assessing the fault detection capability of each test case can utilize all three items above, it still has a deficiency that ignores the similarity between test cases. For mitigating this problem, this paper proposes a hybrid regression test case prioritization technique which aims to achieve better prioritization by incorporating code coverage based clustering approach with BNTCP to depress the impact of those similar test cases having common code coverage. Experiments on two Java projects with mutation faults and one Java project with hand-seeded faults have been conducted to evaluate the fault detection performance of the proposed approach against Additional Greedy approach, Bayesian Networks based approach (BNTCP), Bayesian Networks based approach with feedback (BNA) and code coverage based clustering approach. The experimental results showed that the proposed approach is promising.**

Keywords—Regression testing; Test case prioritization (TCP); Clustering; Bayesian Network (BN)

I. INTRODUCTION

Regression testing is an important but time consuming test activity [1]. Previous research has shown that the time cost of regression testing would account for more than one third of the total time for software maintenance [2]. One of the major concerns in regression testing is to continue improving the rate of fault detection while reducing the cost. In the last few decades, researchers have developed many techniques to make regression testing more efficient. Some of them select a subset of the test suite developed for an earlier version of a software system such as test case selection and test suite minimization techniques. These methods successfully reduce the cost of regression testing but lose a little fault detection capability because some test cases have been discarded during the process. Unlike these two kinds of approaches, test case prioritization can lower the testing cost by executing test cases according to their contributions to testing goals [3] without discarding any test cases. Using this approach, test engineers can adjust the length of the execution sequence of test cases according to their budget.

To date, researchers have paid more and more attention to regression test case prioritization problem and proposed various prioritization techniques [3-10]. Most of them only depend on the code coverage information. Mirarab et al. believed that more information would improve the performance of TCP and constructed Bayesian Networks with source code change information, software quality metrics and test coverage data to prioritize the test cases [4]. The empirical studies demonstrated that their approach had better fault detection rate than those depending on code coverage only. However, they ignored the similarity between test cases which share common coverage. As a result, re-running similar test cases may waste time and it is important to identify and decrease the duplicate execution of similar test cases.

To illustrate this problem, considering a program with a test suite containing three test cases which cover five classes of the program shown in Table I and the corresponding fault coverage matrix shown in Table II, BNTCP may output 1-2-3 as the "optimal" solution. Whereas, the optimal test case ordering for this case is 1-3-2 if the evaluation criterion is APFD (Average Percentage Faults Detected) [3]. BNTCP does not work well in this case mainly because it prioritizes test cases according to their failure probability which represents fault revealing ability, but test cases with common properties (e.g., having similar code coverage) may have similar fault detection ability [5]. Consequently, those test cases with similar code coverage have similar failure probability and thus have similar priority which may reduce the rate of fault detection. Therefore, it is necessary to find a better technique which can overcome this weakness of BNTCP and get better performance in terms of fault detection.

TABLE I. TEST COVERAGE MATRIX

Test Case	Class				
	1	*2*	*3*	*4*	*5*
1	X	X	X	X	
2		X	X	X	
3					X

TABLE II. FAULT COVERAGE MATRIX

Test Case	Fault				
	1	*2*	*3*	*4*	*5*
1	X	X	X	X	
2		X	X	X	
3					X

* corresponding author.

0730-3157/15 $31.00 © 2015 IEEE
DOI 10.1109/COMPSAC.2015.154

This paper proposes a new hybrid regression test case prioritization technique which incorporates code coverage based clustering method with BN based approach for better prioritization. There are two main steps in the new hybrid technique. First, clustering methods are employed to classify all test cases into groups. The test cases in each group share similar code coverage. Second, the clustered test cases are prioritized according to their failure probability by BN based approach. Thus, the hybrid technique will not only make effective use of source code modification information, software quality metrics and test coverage data but also reduce the effects of similarity between test cases.

There are two main contributions of this work: (i) To the best of our knowledge, the work is the first one which integrates the clustering approach to BNTCP. (ii) We use three programs *jtopas*, *xml-security* and *ant* to conduct experiments on our technique and compare it with some related techniques. The empirical results show that our technique is promising.

The rest of this paper is organized as follows. Section II reviews some related work. Section III describes the new approach for TCP. Section IV presents the empirical study. Section V concludes and gives ideas for future work.

II. Related Work

Test case prioritization aims to reduce the cost of regression testing by processing important test cases at an early stage. Rothermel et al. applied Greedy strategy and Additional Greedy strategy to TCP [3]. Li et al. applied two search algorithms including Genetic algorithm and Hill Climbing algorithm to TCP for faster coverage [6]. Jiang et al. proposed a family of coverage-based adaptive random testing techniques for TCP [7]. Similar to [7], Fang et al. proposed a similarity-based TCP technique based on farthest-first ordered sequence [8].

Recently, some researchers used clustering approaches to improve the performance of TCP. Arafeen et al. proposed a clustering approach for TCP that utilized requirements information [9]. Carlson et al. presented a coverage-based clustering approach for TCP [5], which was more relevant to our work. Their approach firstly clustered test cases in terms of code coverage similarity, secondly prioritized test cases within each cluster using several prioritization techniques that utilized code coverage information, a code complexity metric and fault detection history information respectively, and finally generated the complete ordered test suite by selecting test cases from each cluster using a round robin method. To investigate the effectiveness of their approach, they performed empirical studies on Microsoft Dynamics Ax with real bugs. Their results indicated that clustering based approaches performed better than approaches without clustering for fault detection.

Above techniques generally depended on one single source of information. However, one can imply that techniques taking advantages of more data usually have much better performance. Mei et al. proposed a multi-level coverage model to improve the cost-effectiveness of TCP techniques [10]. Mirarab and Tahvildari proposed a prioritization approach based on BN which integrated source code change information, software quality metrics and test coverage data into one unified model [4]. Their approach employed a BN and prioritized test cases according to their failure probability. They also evaluated the performance of their approach using APFD measure on Apache Ant with hand-seeded faults. Their experimental results showed that their approach achieved higher APFD values than coverage based approaches.

Despite the effectiveness of the BN based approach, there remains an issue with this approach. As described in Section I, the BN based approach assigns similar priorities to those test cases with similar code coverage patterns. In this way, the rate of fault detection may decrease because test cases with similar code coverage areas may cover similar faults and test cases covering the same faults make contributions to fault detection only once. Thus, a better test case prioritization technique is needed to overcome the weakness of the BN based technique and to get better performance in terms of fault detection. In this paper, a hybrid technique for TCP is presented which utilizes a clustering approach before constructing BNs to overcome the weakness of BN based technique.

III. Clustering – BN based approach for TCP

In this section, we describe the proposed Clustering – BN based technique for TCP which enhances the BN based approach by clustering test cases before constructing Bayesian Networks. Figure 1 shows an overview of the proposed technique. Our technique contains two main phases. First, test cases are clustered based on their code coverage information. Second, a BN is built to prioritize clustered test cases according to the results of probabilistic inference. The following two subsections describe each phase in detail.

A. Code coverage based clustering

In this step, the agglomerative hierarchical clustering algorithm [11] is employed to cluster test cases. This clustering algorithm uses a bottom-up strategy and groups test cases step by step as follows: First, each test case is treated as a cluster. Second, two closest clusters are merged into one cluster according to the similarity among clusters. Third, the pair-wise similarities between clusters are updated while the new generated cluster is regarded as one single cluster. Finally, the previous two steps are repeated until all the test cases are in one cluster or some terminal conditions are satisfied.

The Euclidean distance on method-level coverage matrix is calculated to represent the distance between test cases. The similarity between two clusters is determined by average linkage strategy. The distance between one test case and a cluster is calculated by averaging distances between the test case and each test case of the cluster. Then take an average of distances between each test case of one cluster and the other cluster.

This algorithm outputs a hierarchical clustering tree that contains all the clustering information from the bottom up. This feature provides an opportunity to adjust the number of clusters which improves the flexibility of the proposed prioritization technique.

B. BN based prioritization

The BN model for TCP designed in [4] is hired in this part.

1) Training process. To present the training process of the BN model, consider a BN containing m C nodes, m correspond-

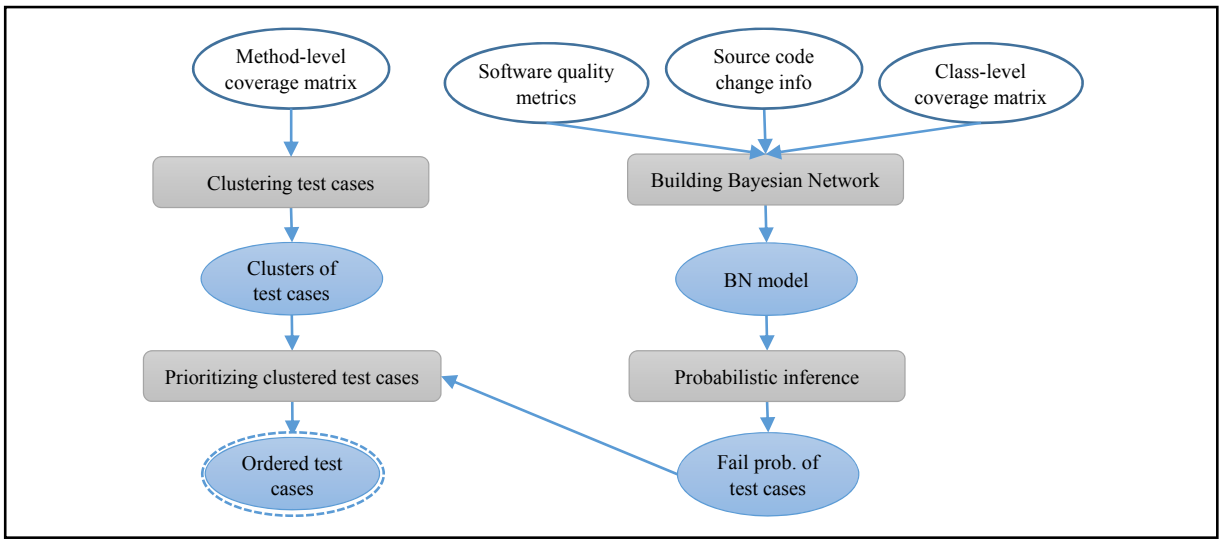


Fig. 1. Overview of our test case prioritization framework.

ing F nodes and n T nodes, as shown in Table III. For each C node, the prior probability $P(c_i)$ is estimated by evaluating the differences of class i between version v and v-1. For each F node, $P(f_i|c_i)$ and $P(f_i|\neg c_i)$ are estimated using two univariate fault-proneness models [12] based on CA (Afferent Coupling) and CBO (Coupling Between Object classes) respectively. CA is a software quality metric that measures the number of classes that depend on a specific class (i.e. uses its methods and/or fields) [13]. CBO is an object-oriented software quality metric of Chidamber and Kemerer (CK) metrics suite [14]. It measures the number of classes to which a given class is coupled (i.e. it uses their methods and/or fields) [14]. For each T node, using the Noisy-OR assumption [15] which assumes that the relations between a test case and its covered classes are independent from each other, we only need to estimate the $P(t_i|f_j)$. In addition, the *leak*, which represents the probability of a test case being failed even though all the classes covered by it are sound, is set as 0.01 in this paper.

2) Probabilistic inference. To estimate the failure probabilities of test cases, the probabilistic inference needs to be performed on the BN that has been built.

3) Prioritizing clustered test cases. To complete prioritization of test cases, first, test cases within each cluster are ranked according to their failure probability. To do so, for each cluster, test cases in this cluster are sorted in descending order of failure probability. Second, test cases are selected from each cluster to generate the final execution sequence of test cases. In this step, we visit each cluster using a round robin method. In each iteration, the first test case in each cluster (if any) is chosen and added into a temporary set (make sure that this temporary set is empty before each iteration). At the same time, the test cases in the temporary set are prioritized with the descending order of failure probability and removed from their clusters. Repeating this iteration process until all the test cases have been added to the final execution sequence (the output). Figure 2 shows an example process of prioritizing clustered test cases. In this example, there are fifteen test cases (T1-T15) which have been divided into five clusters. There are also some two-tuples made up with a test case and its failure probability.

TABLE III. PESUDO CODE FOR ESTIMATING THE CPTS

	Estimating CPTs
	$\alpha + \delta_1 \leq 1$
	$\beta + \delta_2 \ll \alpha + \delta_1 \leq 1$
1	**for** each i $(1 \leq i \leq m)$ **do**
2	$P(c_i) = 1 - similarity(class_i)$
3	$P(f_i\|c_i) = \frac{\alpha\ CA(class_i)}{max(CA)} + \delta_1$
4	$P(f_i\|\neg c_i) = \frac{\beta\ CBO(class_i)}{max(CBO)} + \delta_2$
5	**for** each j $(1 \leq j \leq n)$ **do**
6	**if** $cov(t_j, class_i) > 0$ **then**
7	$P(t_j\|f_i) = cov(t_j, class_i)$
8	**end if**
9	**end for**
10	**end for**
11	
12	**define** function $similarity(class_i)$ **begin**
13	return the similarity of class i between version v and v-1
14	**end**
15	**define** function $CA(class_i)$ **begin**
16	return the CA value of class i
17	**end**
18	**define** function $CBO(class_i)$ **begin**
19	return the CBO value of class i
20	**end**
21	**define** function $cov(t_j, class_i)$ **begin**
22	return the percentage of class i covered by test case j
23	**end**

IV. EMPIRICAL STUDY

In order to evaluate the performance of our proposed technique for fault detection, several empirical experiments have been conducted and they will be described in this section.

A. Research Question

In our experimental studies, we investigate the following research question.

Prioritized Clusters of test cases (after the first step)
Cluster1: (T7, 0.4)-(T3, 0.35)-(T9, 0.25)
Cluster2: (T2, 0.32)-(T6, 0.32)-(T11, 0.28)
Cluster3: (T5, 0.25)-(T8, 0.23)-(T12, 0.2)-(T4, 0.2)
Cluster4: (T15, 0.18)-(T13, 0.15)-(T1, 0.15)
Cluster5: (T10, 0.1)-(T14, 0.05)
Iteration 1
{(T7, 0.4), (T2, 0.32), (T5, 0.25), (T15, 0.18), (T10, 0.1)}
→ T7-T2-T5-T15-T10
Iteration 2
{(T3, 0.35), (T6, 0.32), (T8, 0.23), (T13, 0.15), (T14, 0.05)}
→ T3-T6-T8-T13-T14
Iteration 3
{(T9, 0.25), (T11, 0.28), (T12, 0.2), (T1, 0.15)} → T11-T9-T12-T1
Iteration 4
{(T4, 0.2)} → T4
Ordered test cases (after the second step)
T7-T2-T5-T15-T10-T3-T6-T8-T13-T14-T11-T9-T12-T1-T4

Fig. 2. An example of prioritizing clustered test cases.

How is the performance of the new proposed technique compared to some former techniques for fault detection?

B. Experiment Setup

1) Prioritization techniques. In our empirical experiments, we compare the proposed Clustering – BN based approach (CBN) with Additional Greedy approach (ADD), BN based approach (BN), BN based approach with feedback (BNA) [16] and method-level coverage based clustering approach (MCC).

2) Target objects, Test suites and Faults. Two Java programs (*xml-security* and *jtopas*) with mutation faults and one Java program (*ant*) with hand-seeded faults are employed to evaluate the fault detection performance of the proposed technique with some related techniques. These three open source programs are available on the website of Software-artifact Infrastructure Repository (SIR) [17]. All these programs are assembled with their corresponding JUnit test suites.

To fully evaluate the performance of the proposed technique for fault detection, there should be two types of faults: mutation faults and hand-seeded faults in the object programs. Hand-seeded faults and the corresponding fault matrices are already available in SIR and the fault matrices are constructed using test suites at the test-class level. But only *ant* is employed for the evaluation with hand-seeded faults because *xml-security* and *jtopas* have only a few versions and hand-seeded faults compared to *ant*. Each involved prioritization technique is performed only once on each version of *ant* which contains all available hand-seeded faults. Table IV details the size, the number of test-class level test cases and available hand-seeded faults for each version of *ant*.

To evaluate with mutation faults, the Major Mutation Framework [18] is used to get mutants for *xml-security* and *jtopas*. For each version of these two programs, a mutant pool is generated to produce twenty mutant groups. Each mutant group is made up with five unused and randomly selected mutants that can be killed by test cases in the test suite from the mutant pool. The Major Mutation Framework also provides a mutation analysis back-end that can help us to produce mutation fault matrices for each mutant group. Fault matrices using test suites at the test-method level are generated. Next, the mutated versions for each version will be achieved by adding one group of mutants to the corresponding version. Each involved prioritization technique is performed on each mutated version. The details for each version of *xml-security* and *jtopas* are listed in Table V.

Note that the software quality metrics are generated from the mutated versions and the change analysis is also performed between the mutated versions and the corresponding previous versions.

3) Data collection. In order to perform our proposed prioritization technique, method-level coverage information for clustering test cases and class-level coverage data, source code change information, software quality metrics for building BNs need to be collected.

a) Collecting coverage information. "Cobertura" [19] for Maven is hired to collect the coverage information of the object programs at method-level and class-level in this paper. In a method-level coverage matrix, each element represents whether a method in which one or more lines of code are executed by a test case or not, where "1" represents executed but "0" means not. But in a class-level coverage matrix, each element is a number between 0 and 1, representing the percentage of a class covered by a test case. The interface classes are not considered.

b) Change analysis. To perform change analysis between a version and its corresponding previous version, a tool named "Sandmark" [20] is used here. It is mainly developed for watermarking but it also provides a code differencing function that compares the byte code of Java files and outputs a score of similarity between 0 and 1.

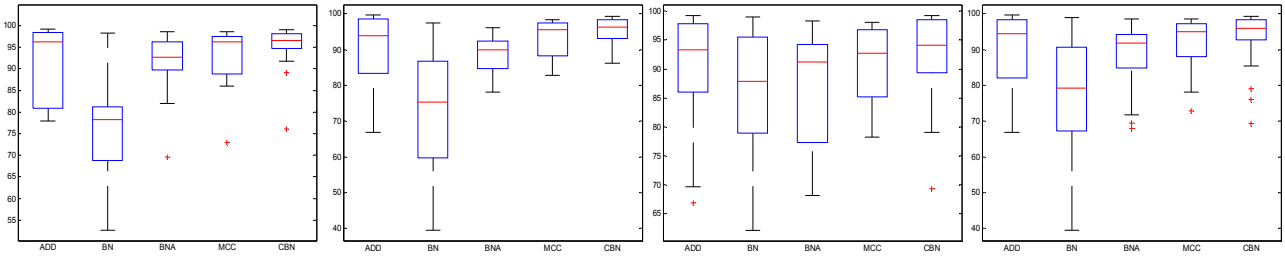
c) Gathering quality metrics. Quality metrics of each class are collected by "ckjm" [21]. This tool calculates Chidamber and Kemerer (CK) object-oriented metrics, CA (Afferent Coupling) and NPM (Number of Public Methods) by processing the byte code of compiled Java files.

4) Evaluation metric. APFD is employed to evaluate the performance of the involved techniques. Higher APFD values mean faster fault detection rates.

5) Parameter settings. Four constants α, δ_1, δ_2, γ ($\gamma = (\alpha + \delta_1)/(\beta + \delta_2)$) that are used to estimate CPTs of F nodes in the BN model are set as 0.8, 0.1, 0.1, 8 respectively. *stp*, a parameter used in the BNA model that controls the number of added test cases in each iteration, is set as 1. k, the number of clusters generated by the agglomerative hierarchical clustering algorithm, is set as 5.

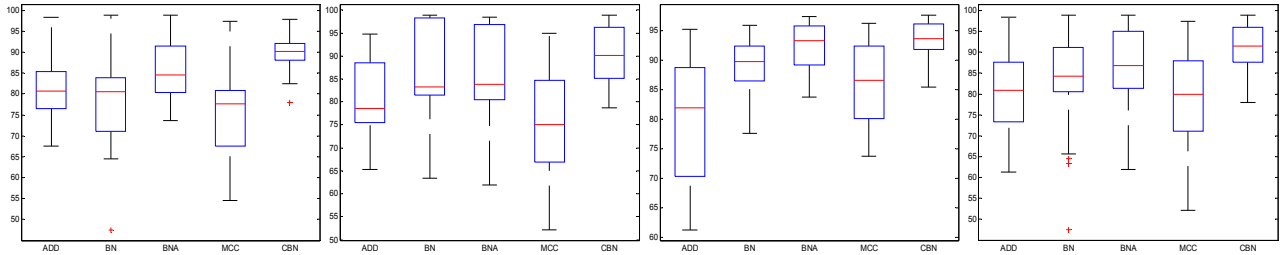
TABLE IV. THE DETAILS FOR VERSIONS OF ANT

Object	KLoC	Number of Classes	Number of Methods	Class-level Tests	Hand-seeded Faults
ant_v1	25.8	229	2511	28	1
ant_v2	39.7	343	3836	34	1
ant_v3	39.8	343	3845	52	2
ant_v4	61.9	533	5684	52	4
ant_v5	63.5	537	5802	101	4
ant_v6	63.6	537	5808	104	1
ant_v7	80.4	627	7520	105	6



(a) *jtopas_v1* (b) *jtopas_v2* (c) *jtopas_v3* (d) *jtopas_all*

Fig. 3. Boxplots of APFD for *jtopas* program.



(a) *xml-security_v1* (b) *xml-security_v2* (c) *xml-security_v3* (d) *xml-security_all*

Fig. 4. Boxplots of APFD for *xml-security* program.

TABLE V. THE DETAILS FOR VERSIONS OF XML-SECURITY AND JTOPAS

Object	KLoC	Number of Classes	Number of Methods	Method-level Tests
jtopas_v1	1.89	19	284	126
jtopas_v2	2.03	21	302	128
jtopas_v3	5.36	50	748	209
xml-security_v1	18.3	179	1627	92
xml-security_v2	19.0	180	1629	94
xml-security_v3	16.9	145	1398	84

C. Results and Analysis

First, we compare the APFD values of the five involved techniques obtained by the experiments conducted on *xml-security* and *jtopas* with mutation faults. The boxplots of the evaluation metric APFD for *jtopas* and *xml-security* are shown in Figure 3 and Figure 4 respectively. Each subfigure indicates the results of one version. The last subfigure in each figure is a summary of boxplot for all versions. Figure 3 and Figure 4 figure out that our proposed approach (CBN) is performed better than the other four approaches including ADD, BN, BNA and MCC. And when compared to *jtopas*, the differences between CBN and the other four approaches are more significant for *xml-security*. Comparing the performances of the other four approaches, we also find that ADD and MCC perform better than BN and BNA on *jtopas* while they perform worse than BN and BNA on *xml-security*. This indicates that these four approaches are not stable compared to our approach CBN.

To investigate the statistical significance of the differences between the five involved techniques, a Friedman test [22] at 0.05 significance level is implemented. The Friedman test is a non-parametric statistical test that can be used to compare group means for statistical significance. The Friedman test rather than Fisher's ANOVA (Analysis of Variance) is chosen because the distributions of each data set obtained by different techniques are not normal and the variances are not all equal. In this case, the "null hypothesis" is that the means of the APFD values for all the techniques are equal. The significance value (*p*-value) will be calculated to decide acceptance of the "null hypothesis".

TABLE VI. FRIEDMAN TEST FOR XML-SECURITY AND JTOPAS WITH MUTATION FAULTS

	Sum of Squares	df	Mean Square	Chi-sq	Significance
Between Groups	305.73	4	76.4323	123.16	.000
Within Groups	885.77	476	1.8609		
Total	1191.5	599			

TABLE VII. MULTIPLE COMPARISON (LSD) FOR XML-SECURITY AND JTOPAS WITH MUTATION FAULTS

Technique (x)	Technique (y)	lower confidence limit	Mean Difference (x - y)	upper confidence limit
ADD	BN	-0.1090	0.4458	1.0007
	BNA	-0.6757	-0.1208	0.4340
	MCC	-0.3548	0.2000	0.7548
	CBN	-2.1423	-1.5875(*)	-1.0327
BN	BNA	-1.1215	-0.5667(*)	-0.0118
	MCC	-0.8007	-0.2458	0.3090
	CBN	-2.5882	-2.0333(*)	-1.4785
BNA	MCC	-0.2340	0.3208	0.8757
	CBN	-2.0215	-1.4667(*)	-0.9118
MCC	CBN	-2.3423	-1.7875(*)	-1.2327
(*) The mean difference is significant at 0.05 level.				

TABLE VIII. APFD MEAN OF THE INVESTIGATED TECHNIQUES FOR ANT

	ADD	BN	BNA	MCC	CBN
All versions	83.3447	81.1909	83.689	84.6392	86.9237
Versions 3, 4, 5 and 7	77.7426	76.5227	77.7956	78.0849	81.3616

If the *p*-value is less than 0.05, the "null hypothesis" should be rejected. Otherwise, the "null hypothesis" should be accepted. The smaller the *p*-value (< 0.05) is, the stronger the evidence will be against the "null hypothesis". The results of the test are exhibited in Table VI.

The results of the Friedman test show that the means of the APFD values for the five techniques are not all equal. This means that the performances of these techniques are significantly different. In order to locate the differences, a LSD (Least Significant Difference) test at 0.05 significance level is launched. It is a multiple comparison method. The results are presented in Table VII. If the interval between lower confidence limit and the upper confidence limit does not include zero, it is believable that the mean difference between the two techniques is statistically significant.

With regarding the APFD values, the results of the LSD test show that the mean differences between our proposed technique CBN and the other four techniques are significant. Moreover, Table VII presents that the mean differences between CBN and the other four techniques are all negative numbers which means that the mean of CBN is larger than the means of the other four techniques. Therefore, it is obvious that CBN significantly outperforms the other four techniques for *jtopas* and *xml-security*, for fault detection.

Second, we compare the means of APFD values of the five involved techniques obtained by the experiments conducted on *ant* with hand-seeded faults, as shown in Table VIII. The table shows that our proposed technique CBN performs better than the other four techniques.

In conclusion, in all the experiments, our proposed technique CBN performs significantly better than the other four techniques including ADD, BN, BNA and MCC for fault detection.

V. Conclusions and Future Work

In this paper, a new regression test case prioritization technique which incorporates Bayesian Networks based approach and code coverage based clustering approach into one framework to improve the fault detection capability has been proposed. This approach is able to enhance the BN based approach and it shows effectiveness in regression test case prioritization problem by clustering test cases before constructing Bayesian Networks. As a result, the proposed approach can achieve full coverage of faults at a faster rate than the BN based approach. After introducing the details of the proposed approach, an experimental study to evaluate the performance of the proposed approach in terms of fault detection has been conducted. The proposed approach has exhibited better performance than four existing approaches in the empirical study.

For the future work, we think that the performance of the proposed approach should be evaluated with more programs and real faults. This will elucidate whether it can maintain the good performance with the practical regression testing process. In addition, more metrics on the relationship between test cases and programs will be emphasized to ensure that the new fault will be exposed as early as possible.

Acknowledgement

This work is partly supported by a project from National Natural Science Foundation of China, with project number '61202030'. The authors also thank anonymous reviewers for their constructive comments.

References

[1] H.K.N. Leung, L.J. White, "Insights into regression testing." In: Proceedings on IEEE International Conference of Software Maintenance, pp. 60-69, 1989.

[2] S. Schach, "Software engineering." Boston, MA: Aksen Associates, 1992.

[3] G. Rothermel, Roland H. Untch, Chengyun Chu, and M. J. Harrold, "Prioritizing Test Cases for Regression Testing." IEEE Transactions on Software Engineering, vol. 27, no. 10, pp. 929-948, 2001.

[4] Siavash Mirarab and Ladan Tahvildari, "A Prioritization Approach for Software Test Cases Based on Bayesian Networks." Fundamental Approaches to Software Engineering, pp. 276-290, 2007.

[5] Ryan Carlson, Hyunsook Do and Anne Denton, "A clustering approach to improving test case prioritization: An industrial case study." 27th IEEE International Conference on Software Maintenance, pp. 382-391, 2011.

[6] Zheng Li, Mark Harman and Robert M. Hierons, "Search Algorithms for Regression Test Case Prioritization." IEEE Transactions on Software Engineering, vol. 33, no. 4, pp. 225-237, 2007.

[7] Bo Jiang, Zhenyu Zhang, W. K. Chan, T. H. Tse, "Adaptive Random Test Case Prioritization." IEEE/ACM International Conference on Automated Software Engineering Proceedings, pp. 233-244, 2009.

[8] Chunrong Fang, Zhenyu Chen, Kun Wu, Zhihong Zhao, "Similarity-based test case prioritization using ordered sequences of program entities." Software Quality Journal, vol. 22, no. 2, pp. 335-361, 2014.

[9] Md. J. Arafeen, Hyunsook Do, "Test Case Prioritization Using Requirements-Based Clustering." 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, pp. 312-321, 2013.

[10] Lijun Mei, Yan Cai, Changjiang Jia, et al. "A Subsumption Hierarchy of Test Case Prioritization for Composite Services." To appear in IEEE Transactions on Services Computing (TSC).

[11] P. Tan, M. Steinbach, and V. Kumar, "Introduction to Data Mining." Addison-Wesley, 2006.

[12] L. Briand, J. Wüst, "Empirical studies of quality models in object-oriented systems." Advances in Computers, vol. 56, pp. 97-166, 2002.

[13] Robert Cecil Martin, "Agile Software Development: Principles, Patterns and Practices." Pearson Education, 2002.

[14] S.R. Chidamber, C.F. Kemerer, "Towards a metrics suite for object oriented design." In: Proceedings of the Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications, pp. 197-211, 1991.

[15] J. Pearl, "Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference." Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[16] S. Mirarab and L. Tahvildari, "An empirical study on Bayesian Network-based approach for test case prioritization." International Conference on Software Testing, Verification and Validation, 278-287, 2008.

[17] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," ESE, vol. 10, no. 4, pp. 405-435, 2005.

[18] R. Just, "The Major mutation framework: Efficient and Scalable Mutation Analysis for Java," in Proceedings of the International Symposium on Software Testing and Analysis, 2014.

[19] Cobertura, available at http://cobertura.github.io/cobertura/.

[20] C. Collberg, G. Myles, and M. Stepp, "An empirical study of Java bytecode programs." Software: Practice and Experience, vol. 37, no. 6, pp. 581-641, 2007.

[21] Ckjm, available at http://www.spinellis.gr/sw/ckjm/.

[22] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance." Journal of the American Statistical Association, vol. 32, no. 200, pp. 675-701, 1937.