A Clustering-Based Test Case Classification Technique for Enhancing Regression Testing

Yulei Pang[1*], Xiaozhen Xue[2], Akbar Siami Namin[2]
[1] Department of Mathematics, Southern Connecticut State University, New Haven, USA.
[2] Department of Computer Science, Texas Tech University, Lubbock, USA.

* Corresponding author. Tel.:+1(203)392-7212; email:pangy1@southernct.edu
Manuscript submitted July 24, 2016; accepted December 23, 2016.
doi: 10.17706/jsw.12.3.153-164

Abstract: To reduce the cost of regression testing, we propose a test case classification methodology based on clustering techniques to classify test cases into effective and non-effective groups. The clustering strategy is based on the coverage information obtained for the earlier releases of the program under test. We employed two common clustering algorithms namely centroid-based and hierarchical clustering. The empirical study results showed the test case clustering can effectively identify effective test cases with high recall ratio and considerable accuracy percentage. The paper also investigates and compares the performance of the proposed clustering-based approach with some other factors including coverage criteria, construction of features, and quantity of faults in the earlier releases.

Key words: Clustering, regression testing, test case classification.

1. Introduction

Regression testing is an expensive yet important activity in software maintenance. In a typical regression testing activity, a software tester endeavors exposing newly introduced or even hidden existing software defects. The test practitioner may focus on the portions of the program that has been affected through the maintenance and refactoring activities. The objective of regression testing is to ensure correctness of the revised program and its behavior after imposing some modifications and changes to the code. It is generally reported that emergence of new and re-emergence of existing but hidden faults is quite common as software is being operated. The defects exposed during regressions testing are mainly observed through two sources: 1) new faults introduced by the modified or added code, and 2) already exiting but hidden faults which have remained unexposed during testing stages performed for earlier releases but revealed for the new release.

A typical regression testing procedure relies on re-executing all or portion of test cases devised for the program under test. A major problem is the cost associated with re-executing a large test suite. It has been reported that the cost of regression testing is as much as two thirds of the overall software life cycle [1]. Therefore, a better and wiser selection of regression tests is desirable to 1) select proper test cases, 2) search for the most cost-effective execution order of selected test cases, and 3) minimize the set of relevant test cases that can be utilized to exercise a newly released version.

This paper proposes classifying test cases into two groups using clustering techniques. The idea is to identify and focus only on effected test cases and thus avoid the needs for re-executing non-effected test cases. When tests cases are clustered into two groups, the test cases in the same group are more similar to each other (in some sense). Therefore, the cluster that contains previously failing test cases, should be given

higher priority of execution. The test case clustering when applied to regression testing can reduce the cost of regression testing substantially [2].

From a statistical point of view, the purpose of cluster analysis is to group a set of objects such that the objects clustered in the same group or cluster are more similar to each other than to those in the other groups or clusters [3]. In practice, there are four categories of clustering techniques, two of which, namely, connectivity-based and centroid-based clustering are suitable for the regression testing problem. More specifically, we apply *k*-means and hierarchical algorithms to cluster test cases. The *k*-means algorithm aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean [4], while the core idea of hierarchical clustering algorithm is to keep objects that are more related to nearby objects together than to objects which are farther away [5], [6]. The contributions of this paper are as follows:

- Introduce a test case clustering-based approach for reducing the cost of regression testing;
- Adopt *k*-means and hierarchical clustering algorithms with test cases' profile information to effectively cluster test cases;
- Evaluate the performance of the proposed technique when a number of factors pertinent to source code coverage, feature construction, and quantity of faults.

The rest of this paper is organized as follows: Section 2 describes the background knowledge of regression testing, and reviews the techniques and algorithms that are referred to in this paper. Section 3 formulates the research problem we aim to address. In Section 4, we evaluate our technique based on a number of experiments. Section 5 provides further discussion. Section 6 presents threats to validity. Section 7 concludes our study and discusses possible future research directions.

2. Background

2.1. Regression Testing

There exist relatively a great number of regression testing techniques such as prioritization-based test case execution, test case selection, and test suite minimization. Various techniques have been proposed to prioritize test cases for regression testing [7]-[10]. The general test case prioritization approach follows two basic key ideas: 1) the utilization of greedy search algorithms with its aims at ranking test cases in a descending order with the hope of detecting newly injected faults in earlier stages of regression testing, and 2) the heuristic-based search approaches to expose remaining and newly introduced faults with the minimum number of test cases selected and to be exercised. Regardless of which search strategy is used, the existing search-based approaches fall short to determine adequacy criteria for regression testing. More precisely, the tester may not be aware when a regression testing process has reached its goal and the testing is adequate enough. Therefore, it is likely that the test practitioner re-run the entire test suite for the newly released version of software system.

The selection-based regression testing techniques intend to determine a subset of a test suit from the previously released software by highlighting affected portions of the code, which have the potential to induce errors. A typical selection technique consists of two major activities: 1) highlight the changed portions of the code, and 2) select test cases which are likely to detect bugs caused by the affected and changed parts. It has widely been reported that use of this technique can reduce the cost of regression testing [11]-[13]. However, several surveys and studies show that very few software industries deploy systematic test selection strategies in their testing activities [14].

Test suit minimization techniques focus on identifying redundant test cases and avoiding their execution with the objective of reducing test suite size and thus reducing the time and effort needed to perform regression test. There exist some research grounded on the assumption that the validation of a specific

requirement can be satisfied by a single test case [15], [16]. However, in practice this assumption is not true because of the complex characteristic of software system, especially for some functional requirements.

2.2. Clustering

As a major approach in data mining and a common technique in statistical data analysis, the most prominent model of clustering algorithms could be categorized into four models: connectivity-based clustering, centroid-based clustering, distribution-based clustering, and density-based clustering. In this section we briefly review two of them, which fit in the regression testing problem very well.

1) Centroid-based Clustering: In centroid-based clustering, clusters are represented by a central vector. *k*-means is one of the simplest yet most popular centroid-based algorithms that are widely used to cluster a set of data points. It is easy to implement and apply this technique even on large data sets and therefore the *k*-mean clustering technique has been successfully applied in various areas, ranging from statistics, data mining, and information technology [3] [4]. Given a set of observations (*x1, x2 , ..., xn*), where each observation is a *d*-dimensional real vector, *k*-means clustering aims to partition the *n* observations into *k* sets *S = {S1, S2, ..., Sk},* with the objective of minimizing the within cluster sum of squares :

$$\sum_{j=1}^{k}\sum_{xi\in Sj}\| xi-uj \|$$

where *uj* is the mean of cluster *Sj*. A typical *k*-means clustering algorithm consists of the following basic steps:

- *Initializing*: Place *k* points into the space to represent the centroid of each group;
- *Clustering*: Assign each object to the group that has the closest centroid;
- *Updating*: Update the centroid of each group;
- *Repeating*: Repeat clustering until the termination conditions are satisfied.

2) Hierarchical Clustering: In data mining, hierarchical clustering is a method of cluster analysis, which seeks to build a hierarchy of clusters. A simple and common strategy for hierarchical clustering generally adopts a bottom up approach [5] [6], so-called agglomerative model, where each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy. Usually the distance between two clusters A and B is measured as the mean distance:

$$\frac{1}{\| A\| \bullet \| B\|}\sum_{x\in A}\sum_{x\in B} d(x,y)$$

$$d(x,y)=\sqrt{\sum_{i}(xi-yj)^2}$$

where *d* is the Euclidean distance. A typical agglomerative clustering algorithm consists of the following basic steps:

- *Assigning*: build clusters for each item, so that if there are *N* items, *N* clusters are constructed, each containing just one item. Let the distances (similarities) between the clusters equal the distances between the items they contain;
- *Merging*: Find the closest (most similar) pair of clusters and merge them into a single cluster, so the quantity of clusters is reduced by one in each iteration;
- *Repeating*: Repeat steps 2 (the merging step) until all items are clustered into a single cluster of size *N*.

The output of the clustering technique is a tree-like structure in which each node represents a cluster.

3. Methodology

3.1. The Problem Formulation

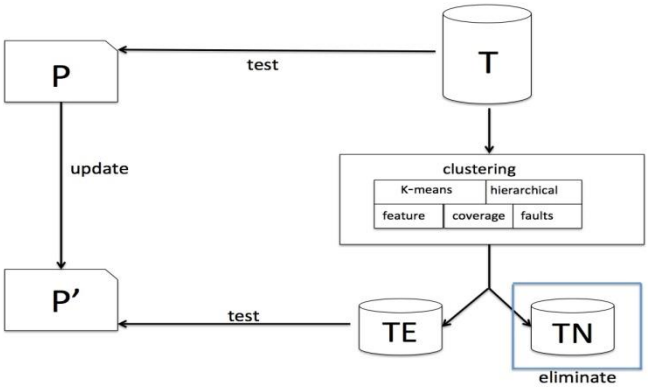

Fig. 1. The proposed clustering-based approach for regression testing.

Suppose there are two consecutive versions released for a given program by fixing the previous faulty statements and updating/changing some executable statements. Given a set of test cases, we aim at classifying each test case into one of the two groups: effective and non-effective test cases. It is our goal to determine effective test cases, exercise only the effective test cases and avoid non-effective test cases and thus minimize the number of test cases that need to be executed for the newly released program. To ease the presentation, let us define a number of terminologies:

- *P* is the program under test.
- *P'* is the modified version of *P*.
- *T = {tc1, tc2, ..., tcn }* is a suit of *n* test cases for *P*.
- *C = {ci}* is the set of statements affected in *P* when upgraded to *P'* where $1 \leqslant i \leqslant \#loc$ where *loc* is the line of codes.
- *TE* is a subset of *T*, which contains effective test cases.
- *TN* is a subset of *T*, which includes non-effective test cases.

As shown in Figure 1, the basic idea of regression test is to re-run test cases in *T* to test *P'*. In terms of the terminologies we just developed, we define test case classification when applied to regression testing as: *Given a program P and its new release P', and a suit of test cases T, initially developed for testing P and will be reused for testing P'; A clustering-based test case classification aims at dividing T into two subsets, i.e. TE and TN such that every defect exposed when P' is executed with T is also detected when P' is executed with TE.*

3.2. The Algorithms

The approach employs Euclidean distance metric along with code coverage information to measure the similarities/dissimilarities between two test cases. Code coverage is a measure used in software structure testing to determine the adequacy of testing. The code coverage based on program statements is the simplest form of this adequacy criterion, which aims at checking whether each executable statement in a given program has been exercised. In the proposed technique, using the binary numeric values 1 and 0 to represent covered/not covered statement; it is possible transform the representation of coverage of statements by each test case to a real value vector. If the Euclidean distance between the vector representations of two test cases is zero then the two test cases are seemingly similar. Similarly, if the Euclidean distance value obtained is some non-zero value, we may assume that the two test cases are different. It is important to note that the magnitude of the Euclidean distance may reflect the significance differences between two test cases and thus their code coverage. A large Euclidean distance indicates that the difference between two test cases is significant. Algorithm 1 and Algorithm 2 describe the procedure of the proposed technique when *k*-means and hierarchical clustering algorithms are adopted, respectively.

Table 1. *k*-Means Clustering-Based Test Case Classification

Algorithm 1 *k*-Means clustering-based test case classification (KMTC).

Require: *T*
Ensure: *TE*, *TN*
1: Initialize *TE*, *TN*
2: Compute *meanTE*, *meanTN*
3: **while** *meanTE*, *meanTN* changed **do**
4: **for** each *tci* in *T* **do**
5: *disTE*=Dis(*tci*,*meanTE*)
6: *disTN*=Dis(*tci*, *meanTN*)
7: **if**(*disTE*> *disTN*)
8: *TE* = *TE* ∨{*tci*}
9: **else**
10: *TN* = *TN* ∨{*tci*}
11: **end if**
12: **end for**
13: Update *meanTE, meanTN*
14: **end while**

Table 2. Hierarchical Clustering-Based Test Case Classification

Algorithm 2: Hierarchical clustering-based test case classification (HBTC).

Require: *T*
Ensure: *TE*, *TN*
1: *C*={}
2: **for** each *tci* in *T* **do**
3: Build a cluster *ci* for *tci*
4: *C*=*C* ∨{*ci*}
5: **end for**
6: **while** there are more than two clusters **do**
7: Find the closest pair of clusters
8: Merge the pair into one cluster
9: Remove the pair of clusters from *C*
10: Add the new cluster into *C*
11: **end while**
12: *ci* = the cluster which contains the previous failing test cases
13: *TE*=*ci*
14: *TN*=*T*/*TE*

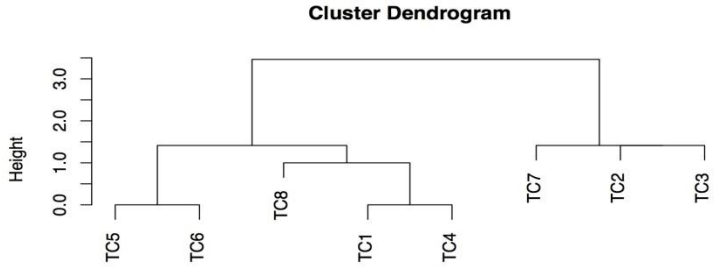

Fig. 2. Results of hierarchical clustering.

3.3. An Illustrative Example

In order to have a better insight of the proposed technique, we present an illustrative example. The code snippet given in Table 3 implements a class to compute sum of the abstract values of two values. Since the

input numbers could be either double or integer, an override method is implemented which can take different input types. The code is composed of 13 lines with one class *AbsSum* along with two methods. A fault is injected on line 11. Eight test cases are devised for the purpose of testing the class and its functionality, and the coverage profile is shown in Table 1 where "–" means non-covered, and "√" indicates the underlying line is covered.

Table 3. An Illustrative Example

		TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8
	a	-1	-1.5	-1.5	-3	1	3	1.5	1
	b	-2	-3.5	3.5	-4	2	4	2.5	-1
1	*public class{*	√	√	√	√	√	√	√	√
2	*public int getAbsSum(int a, int b){*	√	–	–	√	√	√	–	√
3	*if(a<0)*	√	–	–	√	√	√	–	√
4	*a=-a;*	√	–	–	√	–	–	–	–
5	*if(b<0)*	√	–	–	√	√	√	–	√
6	*b=-b;*	√	–	–	√	–	–	–	√
7	*return a+b;}*	√	–	–	√	√	√	–	√
8	*public int getAbsSum(double a, double b){*	–	√	√	–	–	–	√	–
9	*if(a<0)*	–	√	√	–	–	–	√	–
10	*a=-a;*	–	√	√	–	–	–	–	–
11	***if(b<-2)***	–	√	√	–	–	–	√	–
12	*b=-b;*	–	√	–	–	–	–	–	–
13	*return a+b;}}*	–	√	√	–	–	–	√	–
		P	F	P	P	P	P	P	P

Machine learning algorithms often involve building vectors, in which intermediate data are held for further processes. Moreover, the vectors represent a high-dimensional space and hold values for possible features that have been taken into account while performing the classification. We simply use binary value 0 or 1 for the purpose of building read value vectors. Table 3 shows the vectors representing each test case.

By calling Algorithms 1 and 2, we obtain the classification results. The *k*-means clustering divides the test cases into two sets, {1, 4, 5, 6, 8} and {2, 3, 7}, labeled as 1 and 2. The previously failing test cases fall into cluster 2. Therefore, we label all the test cases in cluster 2 as effective test cases. When performing regression testing, the test cases in this cluster will be re-executed. Similarly, cluster 1 holds non-effective test cases. Fig. 2 depicts the results of hierarchical clustering. As shown in Figure 2,, the results of hierarchical clustering and *k*-means clustering are consistent.

4. Experimental Study

4.1. Subject Programs

Table 4 lists the subject programs used for the experimentation. We obtained these extensively used modest sized Java programs, including Nanoxml, Jtopas, Jmeter, XML-security, and ant from the Software Infrastructure Repository [17]. The first two programs are TSL (Test Specification Language) test suits, and the last three are based on Junit test framework. Nanoxml is an XML parser for Java. Jtopas is a small Java library for tokenizing and parsing texts. Jmeter is a Java desktop designed to load test functional behavior and measure performance. XML-security library includes a mature digital signature and encryption implementation. The ant program is a Java library and command line tool whose mission is to drive processes described in build files as targets and extension points dependent upon each other. Table 4 lists the number of versions for each program, the number of class files in the most recent version, the number

of lines of statements in the most recent version, and the number of test cases available for the most recent version.

Table 4. Subject Programs. LOC: Lines of Codes, NC: Number of Classes, NT: Number of Test Cases, NV: Number of Versions

Program	Description	NV	NC	LOC	NT
NanoXML	XML parser	6	26	7,646	216
Jtopas	XML encryption	3	50	5,400	126
Jmeter	Text parser	6	389	43,400	78
XML-security	Load tester	4	143	21,613	83
Ant	Text parser	9	627	85,400	877

4.2. Experimental Setup

To perform test case classification, we require two sets of information: coverage information and initial clustering data (i.e., training data). The *TE* includes all the failing test cases in the previous version, and the *TN* contains just one case, the virtual test case that covers no statements. We obtained coverage information by running test cases on the instrumented subject programs. We instrumented each program by inserting **print** statement into each block to get the converge information. The coverage information obtained for the original program was then used to cluster current version's test cases.

4.3. Evaluation Metric

In the field of statistics and for the classification purposes, four key terms including true positives (*tp*), true negatives (*tn*), false positives (*fp*), and false negatives (*fn*) are usually computed for comparing the results and assessing the performance of the classifier utilized. The terms positive and negative refer to the classifier's prediction, also known as the expectation, and the terms true and false refer to whether that prediction corresponds to the external judgment, also known as the observation [18]. These terms and their associations are illustrated in Table 5 for classification of test cases.

Table 5. Test Cases Classification

	Truly Effective	True Non-effective
Predicted Effective	*tp*	*fp*
Predicted Non-Effective	*fn*	*tn*

Accordingly, three major measurement metrics, i.e. accuracy, precision and recall are usually used to assess how well a binary classification is performed. The accuracy of a measurement system is the degree of closeness of measurements of a quantity to that quantity's actual (true) value. It is the percentage of sum of all true positives and false negatives out of the sum of all the true positives, true negatives, false positives, and false negatives [18].

$$accuracy = \frac{tp + tn}{tp + tn + fp + fn}$$

The precision of a measurement system, also called as *reproducibility* or *repeatability*, is the degree to which the repeated measurements under unchanged conditions show the same results [18]. It is formulated as the fraction of the number of true positives to the sum of true positives and false positives.

$$precision = \frac{tp}{tp + fp}$$

The recall measurement in this context is also referred to as the true positive rate or *sensitivity*, is the ratio of true positives over the sum of true positives and false negatives or the percentage of flows in an application class that are correctly identified [18].

$$recall = \frac{tp}{tp + fn}$$

4.4. Evaluation

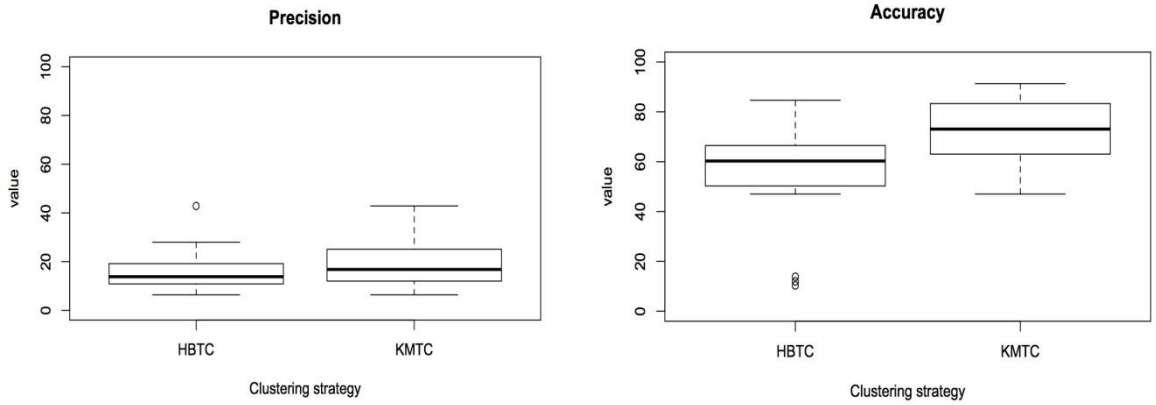

Fig. 2. The performance of the two techniques.

Table 6 reports the performance of the proposed clustering-based approach for classifying regression tests. For *k*-means-based technique, the overall accuracy (73.18%) and recall (100.00%) ratios are considerably good. The precision is measured as 19.32% and it is not so significant when compared to the other two metrics. The hierarchical technique underperforms the *k*-means based technique; similarly the accuracy (53.97%) and the recall (100.00%) ratios are considerably good. The precision is measured as 16.32%. It is important to note that we were not able get the data for ant when hierarchical technique was used, since the failing test cases were always assigned into different clusters. Boxplots in Figure 2 demonstrate and compare the distribution of accuracy and recall for *k*-means and the hierarchical technique.

Table 6 The Performance of the Proposed Techniques

	KMTC			HTC		
	Accuracy	**Precision**	**Recall**	**Accuracy**	**Precision**	**Recall**
nanoXML-V2	78.50%	32.35%	100%	53.52%	11.23%	100%
V3	63.89%	17.89%	100%	35.78%	11.96%	100%
V4	70.37%	30.34%	100%	40.37%	10.34%	100%
V5	81.94%	22.00%	100%	49.21%	13.83%	100%
Jtopas-V2	66.67%	16.00%	100%	66.67%	16.00%	100%
V3	59.52%	15.00%	100%	59.52%	15.00%	100%
Jemeter-V2	82.05%	33.33%	100%	63.89%	17.89%	100%
V3	73.08%	22.22%	100%	73.08%	22.22%	100%
V4	76.92%	28.00%	100%	76.92%	28.00%	100%
V5	60.26%	20.51%	100%	60.26%	20.51%	100%
V6	84.62%	42.86%	100%	84.62%	42.86%	100%
XMLsec-V2	66.27%	6.67%	100%	66.27%	6.67%	100%
V3	62.65%	11.43%	100%	62.65%	11.43%	100%
V4	46.99%	6.68%	100%	46.99%	6.68%	100%
V5	59.04%	10.53%	100%	59.04%	10.53%	100%

ant-V2	91.33%	12.64%	100%	NA	NA	NA
V3	90.88%	14.89%	100%	NA	NA	NA
V4	88.71%	16.81%	100%	NA	NA	NA
V5	86.77%	7.20%	100%	NA	NA	NA
Aveage	73.18%	19.32%	100%	53.97%	16.32%	100%

For the five subjects programs, the recall ratio is 100% demonstrating that we can effectively identify almost all of the actual effective test cases. In other words, the value 100% indicates that all test cases, which can expose a fault, were classified into the effective category by our approach. High accuracy indicates that for most cases, both the actual effective and actual non-effective test cases are classified properly and thus minimizing the cost of regression testing by not running non-effective test cases.

5. Results

5.1. Coverage Criteria

The proposed technique uses coverage information to compute the distance between two test cases. In practice, there are three widely used coverage criteria: function, statement, and block coverage. A research question we would like to investigate is whether achieving a certain level of different coverage criteria will have any significant impact on the performance of our proposed technique? The answer to this question will help us decide which coverage criterion to utilize. To address this question, we conduct further experiments. We set up the experiments following the same steps as discussed in Section 4 base on *k*-means clustering except that the coverage information is measured for function and block criteria in addition to the statement coverage.

Table 7. Performance When the Coverage Criteria are Under Control

Feature	Accuracy	Precision	Recall
Statement	73.18%	19.32%	100%
Block	72.25%	18.59%	100%
Method	58.70%	14.36%	100%

Table 7 demonstrates the effect of different coverage criteria on the performance of our approach. The data is presented in term of the means of overall observations for all the subjects programs. We noticed that different coverage criteria have different impact on the performance of technique. The block coverage criterion is comparable to the statement coverage, where the differences are around 1% on both accuracy and precision. However, both of these coverage criteria are more effective than the function-level coverage criterion, and the difference is significant, i.e. 4% on both accuracy and precision.

5.2. The Choice of Features

In order to adapt clustering technique, features are defined according to the numerical representation used in the statements. In our recent study and in a similar scenario [19] [20], we developed three feature construction methods, called Single (1-mer), Pairwise (2-mer), and 3-Ways (3-mer). The set of pairwise features contained all the possible pairwise sequences of statements. The output for each test case is a *N^2*-dimensional vector where *N* is the total number of statements in the test suits. Similar to the pairwise feature construction, 3-ways check the coverage of each generated 3-way permutation of all the statements. The output for each test case is an *N^3*-dimensional vector. An interesting question arises is whether the way of constructing feature have any influence on the effectiveness of the technique? We developed some Java utility programs to extract the values of each features from the profile files of test output, and re-conducted the experimental study based on one of the subject program, i.e. NanoXML using K-means

clustering.

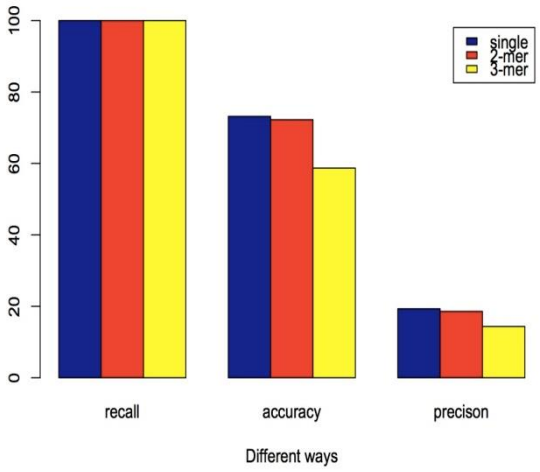

Fig. 3. Bar plots for single, pairwise and 3-mer features.

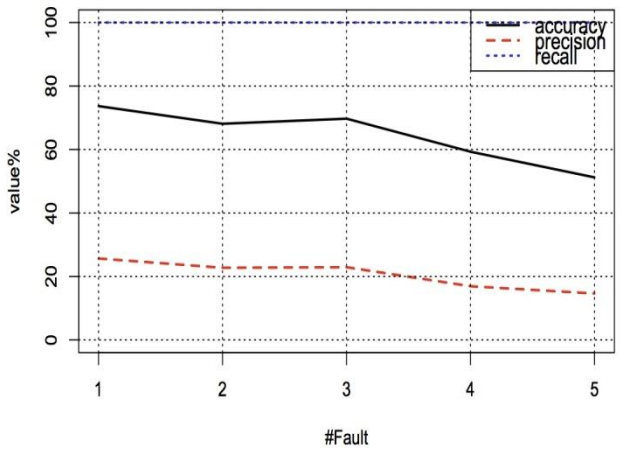

Fig. 4. KMTC for different number of faults.

Table 8. Performance When the Way to Construct Feature is Under Control

Feature	Accuracy	Precision	Recall
single	73.67%	24.66%	100%
pairwise	74.24%	19.99%	100%
3-way	69.62%	17.73%	100%

The experiments results are shown in Table 8 and Figure 3 where data are presented in terms of the means. The performance of single and pairwise are competitive, and the difference of precision and accuracy is no more than 1%, which are negligible. However, both outperform the 3-way case by 4% to 5% in accuracy and by 2% to 8% in precision.

5.3. The Quantity of Faults

Table 9. Performance When the Number of Faults is Under Control

Quantity	Accuracy	Precision	Recall
1	73.67%	25.66%	100%
2	68.11%	22.74%	100%
3	69.72%	22.91%	100%
4	59.30%	16.92%	100%
5	51.21%	14.65%	100%

The faults in the subjects programs are hand-seeded by other researchers and only a single fault exists in any version. However, in practice there are multiple faulty statements in the program under test causing a large number of test cases failing [21]. It is important to investigate the performance of the proposed technique when multiple faults are in present. We instrumented each program with its faults and controlled the activation of the faults incorporated using preprocessors. We repeatedly activated a desired number of faults incorporated in each program. More specifically, we generated faulty versions with k faults by activating simultaneously n instrumented faults.

We re-conducted the experimental study based on one of the subject program, i.e. nanoXML, using K-means clustering. The visualization analysis illustrated in Table 9 and Figure 4 show that overall the accuracy and recall decrease with the increase of quantity of faults.

6. Threats to Validity

Internal Threats relates to the approximate truth about inferences, cause-effect, and confounding relationships among variables as well as the reliability and correctness of the tools utilized in the experiment and the intermediate processes employed while conducting the experiment. In order to get the profile, i.e., coverage information, we instrumented each program in the statement level manually instead of using any tools. Furthermore, we instrumented each block and function instead of statements to capture the statistics.

External Threats relates to the generalization of the results observed through the experiments to larger scales. The experiments in this study are based on some mid-size Java programs containing hand-seeded faults. The hand-seeding faults may introduce some external threat to the validity of experiment.

Construct Threats relates to metrics and measurement and whether they measure the properties that are to be captured. In this study, we utilized the formulas of accuracy, precision and recall to measure the performance of the proposed technique. All the metrics are in the view of classification. In practice, there might be other computation methodologies to assess the performance of regression test cases reduction.

7. Conclusion

We introduced a test case classification methodology based on *k*-means clustering to enhance regression testing. Based on our empirical study we came to the conclusion that the clustering based test case classification can partition test cases with high recall ratio and considerable accuracy percentage. The paper also found out that the clustering-based approach performs better when first the block coverage criterion is utilized, second when single statement or pairwise feature are constructed, and third the performance deteriorated when the number of faults increases. We also observed that for some subject programs failing test cases are always assigned into different clusters thus make it impossible to do binary clustering. One possible solution is to build more than one clusters and further experiments are needed.

References

[1] Pressman, R. (2002). *Software Engineering: A Practitioner Approach*. McGraw-Hill, New York.

[2] Pang, Y., Xue, X., & Namin, A. S. (2013). Identifying effective test cases through k-means clustering for enhancing regression testing. *Proceedings of the 2013 12th International Conference on,Machine Learning and Applications* (pp. 78–83).

[3] Anderberg, M. R. (1973). Cluster analysis for applications. *DTIC Document*.

[4] J. MacQueen., *et al.* (1967). Some methods for classification and analysis of multivariate observations. *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*.

[5] Sibson, R. (1973). Slink: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, *16(1)*, 30–34.

[6] Defays, D. (1977). An efficient algorithm for a complete link method. *The Computer Journal*, *20(4)*, 364–366.

[7] Mirarab, S., & Tahvildari, L. (2008). An empirical study on Bayesian network-based approach for test case prioritization.

[8] Elbaum, S. G., Malishevsky, A. G., & Rothermel, G. (2002). Test case prioritization: A family of empirical studies. *IEEE Trans. Software Eng.*, *28(2)*, 159–182.

[9] Rothermel, G., Untch, R. H., Chu, C., & Harrold, M. J. (2001). Prioritizing test cases for regression testing. *IEEE Trans. Software Eng.*, *27(10)*, 929–948.

[10] ASE 2009. *Proceedings of the 24th IEEE/ACM International Conference on Automated Software*

Engineering.
[11] Harrold, M. J., Rosenblum, D. S., Rothermel, G., & Weyuker, E. J. (2001). Empirical studies of a prediction model for regression test selection. *IEEE Trans. Software Eng.*, *27(3)*, 248–263.
[12] (2009). *Proceedings of the* 25*th IEEE International Conference on Software Maintenan*c.
[13] Rothermel, G., & Harrold, M. J. (1998). Empirical studies of a safe regression test selection technique. *IEEE Trans. Software Eng.*, *24(6)*, 401–419.
[14] Travassos, G. H., Maldonado, J. C., & Wohlin, C. (2006). *Proceedings of the International Symposium on Empirical Software Engineerin*.
[15] Marre, M., & Bertolino, A. (2003). Using spanning sets for coverage testing. *IEEE Trans. Software Eng.*, *29(11)*, 974–984.
[16] Ernst, M. D., & Jensen, T. P. (2005). *Proceedingsofthe 2005 ACM Sigplan-Sigsoft Workshop on Program Analysis for Software Tools and Engineering*.
[17] Do, H., Elbaum, S. G., & Rothermel, G. (2005). Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering: An International Journal*, *10(4)*, 405–435.
[18] Olson, D. L., & Delen, D. (2008). Advanced data mining techniques.
[19] Xue, X., Pang, Y. & Namin, A. S. (2014). Feature selections for effectively localizing faulty events in GUI applications. *Proceedings of the 2014 13th International Conference on Machine Learning and Applications*.
[20] Xue, X., Pang, Y. & Namin, A. S. (2014). Trimming test suites with coincidentally correct test cases for enhancing fault localizations. *Proceedings of the 2014 IEEE 38th Annual*. *Computer Software and Applications Conference*.
[21] Xue, X., & Namin, A. S. (2013). How significant is the effect of fault interactions on coverage-based fault localizations?. *Proceedings of the 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*.

Yulei Pang received her M.S. degree in statistics and her Ph.D. degree in mathematics from Texas Tech University in 2012 and 2014, respectively. She is currently working as an assistant professor in the Department of Mathematics at Southern Connecticut State University, US. Her current research interest includes applied mathematics and applied statistics.

Xiaozhen Xue received his B.S. and M.E. degree in software engineering from Beijing Jiaotong University, China, in 2007 and 2010, respectively. He obtained his Ph.D. degree in computer science from Texas Tech University in 2014. He is currently a software engineer in Amazon, Inc. His current research interest includes software engineering and cyber security.

Akbar Siami Namin received his Ph.D. degree from the Department of Computer Science at the University of Western Ontario, London, Canada in 2009. Currently He is an associate professor in the Department of Computer Science at Texas Tech University. His research interests are Software engineering, testing, and program analysis, cyber security and secure programming.