An Improved Regression Test Selection Technique by Clustering Execution Profiles

Chen Zhang\textsuperscript{1,2}, Zhenyu Chen\textsuperscript{1,2}, Zhihong Zhao\textsuperscript{1,2*}, Shali Yan\textsuperscript{1,2}, Jinyu Zhang\textsuperscript{1,2} and Baowen Xu\textsuperscript{1}
\textsuperscript{1} State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
\textsuperscript{2} Software Institute, Nanjing University, Nanjing, China
\*Corresponding author: zhaozh@software.nju.edu.cn

Abstract—In order to improve the efficiency of regression testing, many test selection techniques have been proposed to extract a small subset from a huge test suite, which can approximate the fault detection capability of the original test suite for the modified codes. This paper presents a new regression test selection technique by clustering the execution profiles of modification-traversing test cases. Cluster analysis can group program executions that have similar features, so that program behaviors can be well understood and test cases can be selected in a proper way to reduce the test suite effectively. An experiment with some real programs is designed and implemented. The experiment results show that our approach can produce a smaller test suite with most fault-revealing test cases in comparison with existing selection techniques.

Keywords—regression testing; cluster selection technique; test suite reduction; fault detection capability

I. INTRODUCTION

Regression testing is performed on modified software to provide confidence that the software behaves correctly and that modifications have not adversely impacted the software’s quality by reusing existing tests [15]. It is absolutely necessary because it ensures that the software quality does not decline during software evolvement. However, the test suite in regression testing is always too large to run all test cases with limited resources. Hence, some strategies have been applied on regression testing to improve the efficiency. A main approach, called regression test selection, is to select a subset of the original test suite, which has enough fault detection capability for the modified codes [7].

Given a modified version of a program, the existing regression test selection techniques mainly focus on how to select test cases which traverse the modifications, called modification-traversing test cases, so that they may reveal the faults. For these techniques, a small subset of test suite is desirable to reduce the cost of regression testing. However, there is a potential risk of omitting test cases that do reveal faults. This trade-off between the time required for selecting and running test cases and the fault detection capability of these test cases is central to regression test selection [7].

Most existing techniques take only one aspect into account. Some focus on test suite reduction. A test suite is reduced to satisfy some test requirements, such as coverage criteria [4]. However, some empirical studies have shown that a subset of test suite satisfying coverage criteria may lose much fault detection capability, though the cost of testing can be reduced dramatically [16]. On the other hand, some researchers have proposed safe selection techniques. That is, all modification-traversing test cases are selected to include all fault-revealing test cases, which can expose faults in the modified program from the original test suite [3, 15]. However, these safe techniques are too conservative, such that the size of the result test suite is still too large for testing resource limit. For example, a safe technique may select 80%~90%, even 100%, of the original test suite but the ratio of fault-revealing test cases is really low [15]. In most cases, if we find most (not all) fault-revealing test cases, it is enough to expose the faults and debug them. This inspires us to improve the precision and reduce the cost of regression testing, despite the unsafe result.

In this paper we provide a scalable solution to trade off test suite reduction and fault detection capability in regression test selection. We propose a new regression test selection technique by clustering execution profiles, called cluster selection technique here. Based on the control flow information of programs, our technique uses cluster analysis to process historical test execution profiles and selects test cases according to the clustering results. An experiment is designed and implemented. The experiment results show that our approach can more significantly reduce the cost of regression testing than existing test selection techniques.

The rest of the paper is organized as follows. Section 2 introduces some related work on regression test selection and cluster analysis in software testing. Our technique is proposed in Section 3 including execution profile collection, cluster procedure and selection strategy. We provide our preliminary experiment and analyze the results in Section 4. Finally, Section 5 concludes the study and looks ahead to our future work.

II. RELATED WORK

Rothermel et al. [14] introduced a definition of safe selection techniques. If a technique can select all fault-revealing test cases from the original test suite, it is safe. They also presented a practical technique for safe test selection [15]. In their technique, all modification-traversing test cases must be selected to guarantee that no fault-revealing test case was omitted. Many similar selection techniques have been proposed in different testing scenes. Chen et al. [3] implemented a tool, called TestTube, to select test cases covering changed entities, like functions, types, macros, in modified programs. Vokolos et al. [19] proposed a safe technique to select test cases by comparing source files of the program and its modified version.
Agrawal et al. [1] and Gupta et al. [8] both based on the dataflow information of programs and selected test cases according to the dataflow modifications. For all these selection techniques, a general evaluation framework was studied [13]. And a few computationally predictors to predict the cost-effectiveness were presented [12]. As a summary, Graves et al. [7] conducted an experiment to study several selection techniques systematically. The goal of this paper is to improve the safe techniques, i.e. to improve the precision of result test suite, despite omitting some fault-revealing test cases.

Some empirical observations have shown that many fault-revealing test cases have some similar behaviors if they come from the same fault [2]. This motivates some researchers to cluster test cases based on some certain metrics. Vangala et al. [18] used program profiles and static execution to compare test cases and applied clustering algorithm on them, identifying redundant test cases with high accuracy. Dickinson et al. introduced cluster filtering technique [5, 6], whose idea was available during the later period of software development when software was stable and some efforts should be made on beta testing. The execution profiles of all test cases were recorded after their execution and cluster analysis was applied on these profiles. Then developers could select a subset of test cases from each cluster with some sampling strategies. Since test cases in the same cluster had similar executions, the subset was representative to find most faults, so it could be used for testing. Ji et al. [10] also used cluster analysis for mutation testing. A huge number of mutants were clustered based on some certain static analysis methods and a small subset of mutants was selected to approximate the original mutant set. Many successful stories of cluster analysis in software testing inspire us to use cluster selection technique in regression testing. In this paper, the cluster analysis method is performed on the modification-traversing test cases. At the beginning of regression testing, the test execution information has been collected for the original versions of programs to assist in identifying test cases, and cluster analysis can be further applied on the information. There is a challenge that whether our cluster selection technique will be a success, i.e. it can also distinguish dissimilar behaviors of the modifications, in regression testing.

III. OUR APPROACH

As mentioned previously, there are some points to be discussed about safe selection techniques. It is certain that a fault-revealing test case is a modification-traversing test case, but not conversely. In many cases, the safe techniques select much more modification-traversing test cases than needed.

We accordingly studied a well-known safe technique presented by Rothermel and Harrold in [15]. We implemented and applied it on a subject program, called space [17]. It is an interpreter for an array definition language, used within a large aerospace application. It has an original version and 38 modified versions, with a test suite of 13585 test cases. Each modified version contains a single fault by modifying one or more lines of code. The subject is well designed, which will be used in the experiment for verifying our cluster selection technique.

We only analyzed 30 of the 38 versions for some reason explained later. For each of these modified versions, we analyzed the result of Rothermel’s safe technique. The main information we cared was the selection precision, i.e. the rate of fault-revealing test cases in the result test set. Fig. 1 shows the selection precision of Rothermel’s technique for each modified version. The bars represent the precision values in percentage terms for all versions. From the figure, it is clear that the selection precision is unfavorable for some versions. We also collected the information of test suite reduction, i.e. the ratio of the selected test suite size to the original test suite size, given in Fig. 2, shown with the same form. As the two figures indicate, for some versions, such as version 1, the selection precision is low while the test suite size is large. Many nonfault-revealing test cases are selected but they actually should be omitted. Since the selection precision is a problem, the test suite can be further reduced by increasing it, on the premise of enough fault detection capability. Thus, we try to find a feasible solution to achieve this goal.

![Figure 1. Selection precision of Rothermel’s technique for space](image)
A smaller test suite could be executed more efficiently and it is more facilitative for test result inspection and test suite maintaining. Even a little decrease on the test suite size will have a considerable effect. So we will improve the regression test selection techniques to reduce test suite further and approximate the same fault detection capability meanwhile.

Our cluster selection technique applies cluster analysis method on the result obtained from safe selection techniques. Cluster analysis is a common technique for statistical data analysis used in many fields. It assigns a set of objects into clusters so that objects in the same cluster are similar in some sense, while objects in different clusters are dissimilar. The rationale for the technique is this: since objects in the same cluster are similar, if we measure the test case executions in some way and use cluster analysis to cluster them, test cases in the same cluster have some similar execution characteristic. If there is a fault in the modified program, the fault-revealing test cases are similar probably because they all reveal the faults, having incorrect executions. After clustering, the fault-revealing test cases are assembled and isolated from most of nonfault-revealing ones because they are not similar. As thus we can easily check each cluster with a predefined strategy and remove a number of nonfault-revealing test cases. The advantage of cluster selection technique over other statistical based approaches is that cluster analysis can group program executions that have similar features, so different program behaviors can be understood more deeply and clearly. Therefore representative test cases can be selected in a proper way to reduce the test suite effectively.

A. Execution Profile Collection

The test selection technique proposed in [15] just mentioned was incidentally implemented as a tool called DejaVu. The process of the tool is this: First, the control flow graphs of the original and modified programs are constructed. Then, the two graphs are traversed synchronously in a depth-first order to identify the modified nodes, which means the corresponding edges are risk. Finally, test cases traversing the risk edges are chosen. The tool DejaVu selects all modification-traversing test cases, which include all fault-revealing test cases. In this paper we will choose Rothermel’s safe technique as our basic technique. Of course any other safe selection technique is also feasible.

At first, the coverage information was collected to form the test history by running all the test cases on the original version of the program. Then we augmented DejaVu and implemented a corresponding tool ourselves. A procedure was added to extract the function execution profiles that characterized the executions of modification-traversing test cases from the test history, after the processing of DejaVu. In this type of profiles, an entry for each function in the program is provided to determine whether or not this function is invoked during the corresponding execution. If the function has been executed, the value is 1, 0 otherwise. Entries for all of the functions constitute a profile instance. There is an instance for the execution of each selected test case and the entries are its 2-value attributes.

B. Cluster Procedure

The inputs to cluster analysis are the profiles just described. After profiling them, all test cases are transformed into metrical form and treated as objects to be clustered. Each of them is represented as a vector, like \( X: <x_1, x_2, ..., x_n> \), where \( x_i \) indicates the execution information for each function, equal to 1 or 0. The number of objects is equal to the number of selected test cases. To imitate the model proposed by Liu et al. [11], we address this step as fingerprinting.

After fingerprinting, the distance between test cases should be calculated for clustering. The distance function we choose is Euclidean distance. The distance is calculated by the formula

\[
D(X,Y) = \sqrt{\sum_{i=1}^{n} d_i^2}, \text{ where } d_i = 0 \text{ if } x_i \text{ is equal to } y_i, 1 \text{ otherwise. (1)}
\]

All attributes of the objects are 2-value, 1 or 0, thus the metric is binary metric in fact. The distance is calculated by applying the Euclidean distance formula on these attributes.
Cluster analysis can be applied on the calculated dissimilarity metric. Simple K-means clustering algorithm is used in our experiment. It is chosen because it is simple and fast, moreover, it performs reasonably well in our preliminary experiments and gives high quality output, which achieves our demand. The discussion about different algorithms is beyond the scope of this paper.

C. Selection Strategy

After performing fingerprinting, distance and clustering functions on the selected test cases, all the test cases are distributed into clusters. And those with similar executions are assigned into the same cluster. If in the initial selected test subset there are several fault-revealing test cases and several nonfault-revealing test cases, the fault-revealing test cases will be assembled and separated from most of the nonfault-revealing ones because they are dissimilar naturally.

There are some conditions that should be noticed. First, although fault-revealing test cases are similar, they may be assigned into more than one cluster because there is still some dissimilarity among them internally. Second, although fault-revealing test cases are different from most of nonfault-revealing ones, they may be assigned into the same cluster because there is some similarity among them. Third, if nonfault-revealing test cases are too few sometimes, which means most of the selected test cases, even all are fault-revealing. All clusters would contain fault-revealing cases so that fault-revealing and nonfault-revealing test cases are not separated obviously.

The selection strategy starts by checking each cluster. We firstly choose a certain proportion of test cases at random, like 2.5%, 5%, 10%, with a minimum of 1 test case, from each cluster. Then each chosen test case is evaluated by comparing the outputs produced by executing both original and modified programs with this test case. As long as the outputs are different, meaning the corresponding test case is found to be a failure, we expect maybe more test cases in the same cluster are fault-revealing since they are similar to this failed test case, therefore we select all the test cases in this cluster as a part of the selection result. If none of the chosen test cases is failed, we expect it is unlikely that any other test case in this cluster is fault-revealing. Therefore we discard all the test cases in this cluster. After all clusters are checked, some clusters are selected while others are discarded, and the final result is generated ultimately.

IV. EXPERIMENT

To obtain our experiment results, we proceed as follows:

1) Firstly we built the augmented DejaVu and applied it on each modified version of each subject program with its original version to select test cases, as well as getting their function execution profiles we would process.

2) Then the execution profiles were clustered by Simple K-means algorithm. The algorithm was implemented under the help of Weka [20]. It provides API to write the clustering procedure.

3) After clustering, we ran the selection procedure we had written. This procedure checked each cluster by choosing a certain proportion of the test cases in the cluster at random and examining them, and finally discarded some clusters in terms of our proposed idea.

4) At last, the useful data was collected and the evaluation model was calculated.

All fault-revealing test cases for each modified version had been known to us by running it with the test suite and comparing the outputs with the original version, just for verification, not a part of our technique. The process of cluster selection technique can be automated and executed any times. We repeated the 3rd processing step by 20 times for each modified version and analyzed the results statistically.

A. Subject Programs

For our experiment we used six C programs, with some modified versions and existing test suites [17]. Most of the programs were studied in previous articles [15]. The programs are listed below:

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>—space</td>
<td>It has been described in the previous section. It can produce good results, which will be studied in detail.</td>
</tr>
<tr>
<td>—a lexical analyzer generator, flex, receiving a lexical rule and outputting a lexical analyzer corresponding to the rule. Different from space, it has more than one original version, and each original version has a modified version which contains more than one seeded fault. For uniformity, we only chose one original version, and regenerated its modified versions by distributing the faults, making one modified version just contain one fault.</td>
<td></td>
</tr>
</tbody>
</table>

—four of the seven C programs collected and constructed initially by Hutchins et al. [9] for use in former experiments, schedule, schedule2, print_tokens, print_tokens2, which were used in many researches, called Siemens Programs. They are tiny, so the number of their different function execution profiles is small. Thus the effect of test selection technique may be little. We made experiments on them just for explaining our technique. In fact, cluster selection technique performs better in large programs like space.

B. Evaluation Models

To evaluate our cluster selection technique, as well as other test selection techniques, we constructed 3 models, one for calculating the fault detection capability, one for calculating the selection precision and another for calculating the test suite reduction. At first, assume that the number of test cases in the original test suite is T, and in a regression testing process T' test cases of them are fault-revealing. Then a test selection technique selects a subset of T' test cases, in which F are fault-revealing.

1) Recall Measure
If the number of fault-revealing test cases in a regression testing process is F', safe techniques could select all of them since they select all modification-traversing test cases. Please note that cluster selection technique is not safe because of the indeterminacy of clustering and randomness of selection
strategy. As mentioned previously, it is enough to expose the faults as long as we are aware of the majority of fault-revealing test cases. Therefore, this can be tolerated for significantly reducing the size of test suite. We provide the recall measure to indicate the fault detection capability. If in a certain run the technique finds $F_T$ fault-revealing test cases, the recall can be calculated as this:

$$\text{Recall} = \frac{F_T}{F_r} \times 100\%.$$  \hfill (2)

It is clear that the recall of safe techniques is 100%.

2) Precision Measure

If in a certain run the technique selects a subset of $T'$ test cases, in which $F_T$ test cases are fault-revealing. The precision of test selection can be calculated as this:

$$\text{Precision} = \frac{F_T}{T'} \times 100\%.$$  \hfill (3)

Larger precision value means smaller test suite and more fault-revealing test cases relatively, which is our goal.

3) Reduction Measure

If an original test suite contains $T$ test cases and a subset of $T'$ test cases is selected by some technique, the test suite reduction model can be calculated as this:

$$\text{Reduction} = \frac{T'}{T} \times 100\%.$$  \hfill (4)

Note that the reduction measure means the ratio of the selected test suite size by any test selection technique to the original test suite size.

4) The relation of three measures

Assume that there are two techniques $t_1$ and $t_2$. After applying them in a regression test selection process, the recall of them are $r_1$, $r_2$, the selection precision are $p_1$, $p_2$, and the test suite reduction are $red_1$, $red_2$, respectively. Then we can get an equation by some simple manipulation using (2), (3) and (4):

$$\frac{r_1}{r_2} = \frac{red_1}{p_1} \times \frac{p_2}{red_2}.$$  \hfill (5)

It is clear that if $t_2$ is a given basic safe technique, the values of $r_2$, $p_2$ and $red_2$ are fixed. On the other hand, if $t_1$ is cluster selection technique, the trade-off between test suite reduction and fault detection capability means $r_1$ need be large enough even if it may not reach 100%, while $red_1$ need be decreased much more. Therefore, from the equation, to make cluster selection technique beat the basic one, $p_1$ need be large, which means relatively high precision.

C. Recall Expectation

Since the selection procedure in cluster selection technique chooses test cases at random, the results of selection may be different. For evaluating the confidence of our cluster selection procedure, we will propose a formula to calculate the recall expectation by the procedure. Assume that there are $k$ clusters after clustering. For any cluster $i$, there are $m_i$ test cases, in which $f_i$ fault-revealing test cases and $n_i$ non-fault-revealing test cases. In the selection procedure, assume that $p\%$ of the test cases are chosen in each cluster for checking. We define $c_i$ as the round of $m_i \times p\%$, with a minimum of 1. Then the recall expectation ($RE$ for short) is calculated as follows:

$$RE = \sum_{i=1}^{k} ((1 - \frac{A_i^{c_i}}{A_m^{c_i}}) \frac{f_i}{\sum_i f_i}).$$  \hfill (6)

$A_i^{c_i}$ means that selecting $r$ from $n$ elements to form a permutation, equal to $n^* (n-1)^* ... (n-r+1)$ or $n! / r!$. For example, $A_{10}^4 = 10 * 9 * 8 * 7$. The meaning of the formula is this: for cluster $i$, we choose $c_i$ test cases and then $A_i^{c_i} / A_m^{c_i}$ is the probability of selecting none of the fault-revealing test cases in cluster $i$. Therefore, $1 - (A_i^{c_i} / A_m^{c_i})$ is the probability of selecting at least one fault-revealing test case in cluster $i$, i.e., the probability of preserving cluster $i$ according to the selection strategy. Then multiply the probability value by the partial recall for cluster $i$, i.e. the ratio of the number of fault-revealing test cases in cluster $i$ to the total number of fault-revealing test cases. This product for cluster $i$ is 0 if $f_i$ is 0, i.e. there is no fault-revealing test case. Accumulating the product over all clusters finally forms the recall expectation. It is clear that the sum increases as $p\%$, or $c_i$ increases.

![Figure 3. Recall expectation of space for different $p$ values](image)

- 2.5%: 81.95% (avg)
- 5%: 88.42% (avg)
- 10%: 92.72% (avg)
- 20%: 96.13% (avg)
Fig. 3 shows different recall expectation corresponding to different $p\%$ values the procedure chooses, for each version of space program. And the average expected recall values over all versions on different $p\%$ are also given in the figure. As $p$ increases from 2.5 to 20, the recall expectation increases consequently. From the figure, it is rational to choose 5\% of the test cases in each cluster, because for most of the versions the recall expectation is large enough, with only a small part of test cases chosen for checking. Although the recall expectation will be larger as the $p\%$ increases, the efficiency of the selection procedure will be affected since more test cases need be executed for checking. The result test suite would also be larger, so that more cost will be consumed in regression testing. Since the expected recall is already large enough for us to expose the faults and debug them, it is relatively proper to choose 5\% for space. For the 5 other subject programs in the experiment, we also choose this value. Please note that this is just an empirical value of the sampling ratio.

![Figure 4. Recall of the two techniques for space](image)

![Figure 5. Selection precision of the two techniques for space](image)
D. Experiment Result

Fig. 4 uses boxplots to depict the recall result of cluster selection technique for each of the modified versions of space program. (The recall of the basic safe technique is not given because it is 100% clearly.) Each boxplot represents the statistics of the recall measure of our technique for each version. As mentioned previously, we repeated the selection procedure by 20 times over all versions. In each one of the boxplots, the height of the box spans the central 50% of the 20 calculated recall values, and its upper and lower end mark the upper and lower quartiles. The line within the box denotes the median of the values in the box and the point denotes the mean. The T-shaped whiskers indicate the 10% and 90% quartiles respectively. We only analyzed 30 versions because for some versions there was no fault-revealing test case and for other versions there were too few test cases initially selected by the basic technique, thus it was not necessary to use cluster selection technique. As the figure indicates, nearly over all of the versions, our technique could statistically select a large part of fault-revealing test cases, more than 80%. Therefore, it means that our technique could select most of fault-revealing test cases so that the fault can be easily found and debugged.

Fig. 5 shows the result on the selection precision of the two techniques, for each modified version. The result of the basic technique is given once again for comparison. The precision measure indicates the ratio of useful test cases in the selected test suite. Each boxplot represents the statistics of the selection precision of our technique for each version and the corresponding bold dot represents the selection precision of the basic safe technique. The meaning of the boxplot is the same as Fig. 4. It is clear that the gap between each pair of the boxplot and the bold dot is the improvement of our technique over the basic one. We can see our technique significantly increases the selection precision, which means it removes a lot of nonfault-revealing test cases. From (5), it is supposed that for each version the test suite is reduced more than the basic one. And Fig. 6 confirms this inference, in which similar boxplots are used to represent the result test suite size, i.e. the reduction measure by our technique, and the bold dots represent the result by the basic one. The difference in test suite size means the test suite is smaller by our technique. Please note that for some versions, the reduction measure for the two techniques are both relatively small so that the gap may not be obvious, but in fact, the absolute test suite size is decreased effectually in proportion.

In summary, for Fig. 4, 5 and 6, the benefit of cluster selection technique is palpable. Almost for all versions, most of fault-revealing test cases are picked out with large recall values. On the other hand, higher precision indicates that the test cases we select to make up final test suite are much less than the result of the safe selection technique. Take version 1 as an example again, after using cluster selection technique, the recall is more than 80%, while the selection precision reaches about 25%. And the test suite size is decreased so significantly. Although the technique discards some fault-revealing test cases in fact, the test suite is reduced more materially, so it could improve the efficiency of regression testing by lowering the consumption of many resources. Overall, our technique provides an effective and efficient approach. Now the trade-off between test suite reduction and fault detection capability is found out.

In these figures, there are still some issues to be illustrated. For some versions, such as version 2, the precision of the basic safe technique is already high enough, close to or equal to 100%. Since the nonfault-revealing test cases are too few, according to the procedure of cluster selection technique, it is certain that the precision will remain high by our technique and the recall will also be large. On the other hand, for some versions, such as version 23, the precision of the safe technique
is very low. Our technique might have difficulty in finding the fault-revealing test cases, so the recall could be low sometimes. Although it is not always the case, we still pay attention to it and try to solve this problem in the future work.

In a similar way, Fig. 7, 8 and 9 respectively show the same related information of cluster selection technique for the other programs, over all modified versions of each program. However, we just give the results on average in the form of histogram for briefness and simplicity. The average results can also explain the benefit. Due to limitations in the base tool used in Rothermel’s fundamental technique, we only analyzed 10 of the modified versions in flex because the structure of this subject and other artifacts in it were not constructed for fitting the tool well. We also neglected 2 versions in print_tokens and 1 version in schedule2 since there was no fault-revealing test case for these versions.

Figure 7. Recall of the two techniques for 5 programs

Figure 8. Selection precision of the two techniques for 5 programs

Figure 9. Test suite reduction of the two techniques for 5 programs

In Fig. 7, the bars represent the recall of our technique for these programs. In Fig. 8, the bars are all divided into two parts: the grey part represents the precision of the basic safe technique and the whole represents the precision of our technique. That is the white part is the improvement of our technique over the basic safe technique. In Fig. 9, conversely, the grey part of the bars represents the result test suite size by our technique, while the whole represents the same information by the basic safe technique.

From these figures, we can get similar expected results, though they are not so effectual for those tiny Siemens programs. After processing, for each program, the recall is large enough with a small test suite, and the selection precision is increased, therefore the test suite is high-caliber and the cost of regression testing will be decreased. In fact, although things are different over all the programs and versions, the results are satisfactory whether for each modified version or on average.

The time required to select tests using our cluster selection technique and run them is also recorded, which indicates the efficiency. In fact, the time of our technique, including clustering time and selection time, is little. Comparing with Rothermel’s safe technique, the time of clustering and selection is extra, but the size of test suite is reduced significantly by our technique, so the run time decreases more and overall the total time becomes much less accordingly. In addition, many other efforts and resources put into regression testing will also be saved. Although sometimes the precision of the basic technique is already high so that there is little improvement using our technique, it is still excellent in general.

E. Threats to Validity in the Experiment

The experiment has the following limitations that may affect the validity. Threats to external validity include the use of only a small set of subject programs, modified versions and test suites. In practice, the situations can be challenging. However, we think that the subject programs are already representative of real programs since they have been studied in many researches. Threats to internal validity include the
correctness of collecting execution information, selecting test cases and clustering processes. We rely on Weka for clustering, the revised DejaVu tool for selecting test cases, and minimize other threats to internal validity by inspecting our data-collection codes and results carefully.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed a new regression test selection technique called cluster selection technique. We introduced the idea, explained its rationale and experimentally verified its effect in comparison with safe selection techniques. We reported the results by means of some constructed models. The experiment results suggest that cluster selection technique can reduce the size of test suite significantly, on the premise of finding most of fault-revealing test cases. Therefore, the cost of regression testing could be further decreased with enough fault detection capability. This technique effectively deals with the trade-offs between test suite reduction and fault detection capability, performing better on large programs.

There are several promising directions for our technique. First, since the experiment results are encouraging, more experiments should be planned and made, for more basic techniques, programs and test suites, to support the theory. In addition, some variables, like the proportion of test cases chosen in selection strategy, should be discussed. Different values of these variables would be expected and investigated. Second, the measurement of programs, function execution profiles, may be simple. It impacts the results. Different results may be obtained using different kinds of measurement. For example, we can record the number of times that functions are called during the corresponding execution, or we can record the execution profiles for statements. We will consider these issues, and verify them by further experiments, no matter whether the result is better or not. Finally, the clustering algorithm can also be changed. We will try other algorithms to get more empirical results and compare them to find a most appropriate algorithm working in our technique.

REFERENCES