2011 Fourth IEEE International Conference on Software Testing, Verification and Validation

Using Semi-Supervised Clustering to Improve Regression Test Selection Techniques*

Songyu Chen[1,2], Zhenyu Chen[1,2,§], Zhihong Zhao[1,2], Baowen Xu[1], Yang Feng[1,2]

[1] State Key Laboratory for Novel Software Technology, Nanjing University, China

[2] Software Institute, Nanjing University, China

[§]Corresponding author: zychen@software.nju.edu.cn

ABSTRACT

Cluster test selection is proposed as an efficient regression testing approach. It uses some distance measures and clustering algorithms to group tests into some clusters. Tests in a same cluster are considered to have similar behaviors. A certain sampling strategy for the clustering result is used to build up a small subset of tests, which is expected to approximate the fault detection capability of the original test set. All existing cluster test selection methods employ unsupervised clustering. The previous test results are not used in the process of clustering. It may lead to unsatisfactory clustering results in some cases. In this paper, a semi-supervised clustering method, namely semi-supervised K-means (SSKM), is introduced to improve cluster test selection. SSKM uses limited supervision in the form of pairwise constraints: Must-link and Cannot-link. These pairwise constraints are derived from previous test results to improve clustering results as well as test selection results. The experiment results illustrate the effectiveness of cluster test selection methods with SSKM. Two useful observations are made by analysis. (1) Cluster test selection with SSKM has a better effectiveness when the failed tests are in a medium proportion. (2) A strict definition of pairwise constraint can improve the effectiveness of cluster test selection with SSKM.

Keywords

Test selection, regression testing, semi-supervised clustering, pairwise constraint, K-means.

1. INTRODUCTION

Regression testing is performed on modified software to provide confidence that the software behaves correctly and modifications have not adversely impacted the software quality by reusing existing tests [21]. Since new faults will be potentially introduced during the modification of programs, regression testing plays an important role in the lifecycle of software development and maintenance [26,27].

However, regression testing is an expensive process due to the fact that the test set increases during the iterative development process. Running the whole test set is an unacceptable strategy in practice because the testing will exhaust the limited resources. An alternative strategy, test selection, is proposed to choose a small but effective subset of the original test set [13, 19, 28, 29, 34].

In recent years, cluster test selection is proposed as an efficient method in regression testing and observation-based testing [17, 18, 32, 34]. Cluster test selection groups tests into some clusters through establishing some dissimilarity (distance) measures. The distance measures are based on the historical execution profiles of tests. Tests with similar profiles will be grouped into same clusters. It is expected that tests which can detect same faults can be put together. And then, a certain sampling strategy is used to select the representative tests from each cluster [32]. Clustering plays a key role in the process of cluster test selection. A good cluster result can well distinguish passed and failed tests, such that we can construct a small test set which has strong fault detection capability. This inspires us to improve cluster results, as well as test selection results.

In machine learning, clustering is an unsupervised learning of a hidden data concept [7]. This technique uses unlabeled data for training and organizing data into clusters. As an unsupervised data analysis procedure, the clustering results depend on the input parameters, such as the number of clusters or the seeds of data [8]. On the other hand, labeled data is useful for the learning results. Labeled data is often limited and expensive to generate, since the labeled information typically requires human expertise and domain knowledge [6,10,12]. In some applications, labeled information can be extracted from partial results or some evidences in advance, such that both unlabeled data and labeled data are available in the applications. Hence the clustering methods using both unlabeled data and labeled data have attracted many researchers. These methods are so-called semi-supervised clustering [1, 2, 5, 6, 9-12, 14, 16].

Semi-supervised clustering, which uses limited amounts of supervision in the form of labels on the data or constraints among data, has become a cutting-edge technique of cluster analysis [9, 10]. The labeled information is often generated by the domain experts or from some evidences. Semi-supervised clustering could improve the precision of clusters in most cases, although Basu et al. figured that some inappropriate labeled data might degrade the effectiveness of clustering because this type of information would mislead the process of clustering [10].

A main challenge of semi-supervised clustering for test selection is to derive appropriate information from previous test results to guide the clustering process. The previous test results are analyzed to form some constraints. These constraints can help us get some labeled data further. Therefore, there is a potential chance to use semi-supervised clustering methods to improve test selection in regression testing. This part needs manual intervention to give

* The work described in this article was partially supported by the National Natural Science Foundation of China (90818027, 60803007, 61003024), the National High Technology Research and Development Program of China (863 Program: 2009AA01Z147), the Major State Basic Research Development Program of China (973 Program: 2009CB320703).

978-0-7695-4342-0/11 $26.00 © 2011 IEEE
DOI 10.1109/ICST.2011.38

constraint definitions. The results of semi-supervised clustering will be affected by these definitions in a large extent.

In this paper, we propose an improved test selection method using semi-supervised clustering. The semi-supervised clustering method used here is semi-supervised K-means [35], or SSKM for short. SSKM uses pairwise constraints to aid unsupervised learning. The constraints can be generated automatically or manually. In our approach, a few of tests will be selected randomly and executed firstly. The test results, such as pass or fail, will be used to derive some pairwise constraints by different definitions. Two types of pairwise constraints, Must-link and Cannot-link, are used in this paper. An efficient algorithm SSDR [35] is adopted to transform original data into a new distance space, such that the clustering results by K-means are better than before. An experiment is designed and conducted on some subject programs from SIR website [36]. The experiment results show that the semi-supervised clustering SSKM can improve test selection significantly.

In summary, this paper mainly makes the following contributions:

- To the best of our knowledge, it is the first time that semi-supervised clustering is used in test selection, even in software testing. This successful story will inspire more researchers to adopt novel machine learning techniques in software testing.
- A comprehensive analysis of experiment results is conducted and some useful observations are made. (a) Cluster test selection with SSKM has a better effectiveness when the failed tests are in a medium proportion. (b) A strict definition of pairwise constraint can improve the effectiveness of cluster test selection.

The rest of this paper is structured as follows. In Section 2, we present some related work on cluster analysis in test selection and some existing semi-supervised clustering methods and their applications. In Section 3, we describe the SSKM method for test selection in detail. A simple example is used to illustrate our technique. In Section 4, we provide the detailed design and result analysis of the experiment. In Section 5, we draw the conclusion and figure out some directions in future work.

2. Related Work

Many regression test selection techniques have been proposed and published in recent decades [19]. One type of them selects the tests which cover the modified software parts with different granularities, such as statements, functions or modules. A typical technique, namely safe test selection, was proposed by Rothermel et.al and implemented as a tool DejaVu [27]. First, the DejaVu tool constructs the control flow graphs of original and modified programs. Then it traverses these two graphs synchronously and identifies the modified nodes and edges. Tests traversing the modified edges will be selected to build up a new test set. Some researchers used the specifications or metadata of software instead of the source code of software. Sajeev et al. [29] used UML specifications and Orso et al. [26] used metadata in XML format in their test selection techniques. Harrold et al. [23] proposed a heuristic strategy by grouping tests to cover modification of software. Black et al. [4] used two coverage criteria, namely Bi-criteria model, to improve the diversity of test selection and reduction. These techniques do not use cluster analysis to assist test selection.

Tests detecting a same bug always have some hidden connections. Dickinson et al. used cluster test selection to mine the hidden connection based on their execution profiles [17]. Masri et al. conducted an empirical study to show the effectiveness of cluster filtering technique [25]. Zhang et al. enhanced Rothermel's safe selection technique by clustering the execution profiles of modification traversing tests [34]. Duan et al. improved cluster test selection using program slicing for dimensionality reduction [18]. Shin et al. proposed a cluster-based test case prioritization technique to reduce the number of pair-wise comparisons [33]. However, the clustering methods mentioned above are unsupervised. The previous test results are not used in the clustering process.

Bowring et al. used active learning for test classification [4]. The main difference between active learning and semi-supervised learning is that the former is a manual iteration process and the latter completes the process automatically after the objective function has been established. Active learning needs an expert to mark the data when the algorithm proposes a classification plan actively. In an active learning framework, it aims to obtain a better partition of the data with minimal number of queries [10]. Both Davidson et al. [16] and Wagstaff [31] show that if queries are not selected properly, then the effectiveness and performance of learning may be reduced in active learning.

There exist many semi-supervised clustering methods and their applications. Zhang et al. and Tang used pairwise constraints to project original data into new perspective to achieve semi-supervised clustering [30,35]. Demiriz et al. used metric learning technique based on an adaptive distance measure in the clustering process [14]. Basu et al. unified the two different methods as a hybrid method under a general probabilistic framework [6]. Semi-supervised learning methods are used in a lot of practical application scenes, such as graph distinguishing, literature, intrusion detection, medical research, etc. [1,2,20]. As far as we know, this is the first time to use semi-supervised clustering for test selection.

3. Our Approach

This section presents our test selection approach. First, we give a framework to introduce the procedure of our approach. Then a semi-supervised clustering method SSKM, including some important formulas, will be explained in detail.

3.1 Framework

As shown in Figure 1, test selection using (unsupervised or semi-supervised) clustering mainly contains four steps:

1) **Capturing Feature:** In regression testing, the results and execution profiles of tests will be recorded. In this paper, a simple execution profile, *function call profile*, is generated for each test. The function call profile is represented as a binary vector, in which each bit records whether the corresponding function is called or not in a running test. If a function is called, the corresponding bit value is set to 1, otherwise 0.

2) **Distance Measure**: For each pair of tests, we will calculate the distance of them. In this paper, we use *Euclidean distance* as the dissimilarity measure. If the binary vector of two profiles are $x: <f_1, f_2, ..., f_n>$ and $x': <f'_1, f'_2, ..., f'_n>$ (n is the number of function) , the distance between them will be:

$$D(x, x') = \sqrt{\sum_{i=1}^{n} (f_i - f'_i)^2} \qquad (1)$$

The attributes f_i and f'_i represent whether the ith function is executed in test x and x'. If it is executed by test x then f_i=1, otherwise f_i=0.

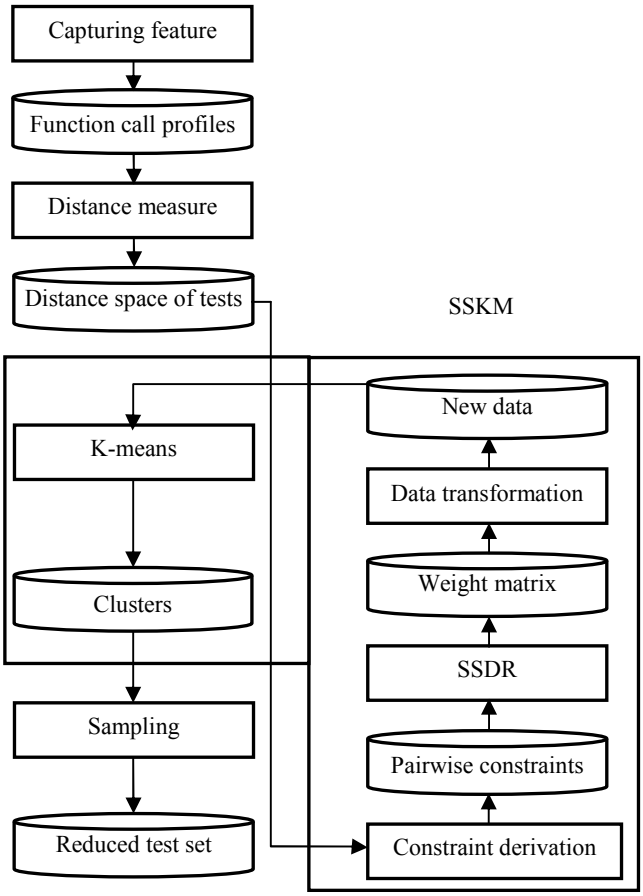

Figure 1. The framework of our approach. The procedure of semi-supervised clustering SSKM is depicted in the textbox.

3) **Cluster Analysis**: The inputs to cluster analysis are the *function call profiles* generated in the first step. In current machine learning research, K-means is one of the most widely used clustering methods. It is efficient and easy to implement, moreover it performs reasonably well in preliminary experiments and gives high quality outputs [32,34]. Instead of K-means, the semi-supervised K-means (SSKM for short) is introduced to improve the clustering results in this paper. And the improved test selection results by SSKM will be compared with the results by K-means.

4) **Sampling Strategy**: A number of sampling strategies can be used to select a few of tests from each cluster and build up a new test set for regression testing. There are some common used strategies, such as n-per-cluster, adaptive sampling [17], dynamic sampling [32], etc. In this paper, adaptive sampling strategy is adopted. A few of tests is randomly sampled from each cluster by a certain percentage. If any selected test is fail, all tests in this cluster will also be selected to build up a new test set.

3.2 Semi-supervised clustering

K-means clustering (MacQueen, 1967) is one of the simplest and most common used unsupervised learning methods. It automatically groups data instances into k clusters so as to minimize the value of objective function:

$$J_{kmeans} = \sum_{j=1}^{k} \sum_{x_i \in C_j} || x_i - \mu_j ||^2 \qquad (2)$$

In the formula above, x_i is an instance in a cluster C_j, u_j is the mean of instances in the cluster C_j. j is from 1 to k, where k is the number of clusters. At first K-means selects k instances randomly from the set that are being clustered. These k instances represent initial cluster centroids. Then it refines them iteratively after new instances are assigned to the clusters. This iterative process of K-means clustering is to achieve maximum similarity in the same clusters (internal objective) and maximum dissimilarity between clusters (external objective) [7]. The algorithm is convergent when the assignment of instances to clusters has no further change. Because of its quality and efficiency, K-means is one of the most popular clustering methods in scientific research and industrial applications. However K-means is sensitive to the selected k initial cluster centroids. It may be trapped in local optimum sometimes. Therefore many researchers are trying to overcome the sensitivity of K-means [22].

The goal of clustering in this paper is to group tests, which detect a same fault, into a same cluster [17]. However the clustering results may not be satisfactory, because the execution profiles cannot capture software behaviors accurately. That is, the tests revealing a same fault may be assigned to different clusters, and the irrelevant tests may be grouped into a same cluster. To improve the clustering results, we introduce a semi-supervised clustering method namely semi-supervised K-means (SSKM). We combine SSDR [35] and K-means to implement SSKM in this paper. SSDR is used as a pre-process of the original data with limited constraint information before K-means.

In the process of semi-supervised clustering, the pairwise constraints are used to express the relationships between instances. The pairwise constraints are more practical than other labeled information, such as cluster labels. In many cases, we can easily specify whether two data instances belong to a same cluster or not. Fortunately the pairwise constraints can sometimes be automatically generated [10]. Therefore, we use pairwise constraints derived from previous test results to improve cluster results in this paper.

In this paper, we use two types of pairwise constraints to specify the relationships. (1) Must-link: two data instances must be assigned to a same cluster. (2) Cannot-link: two data instances must be assigned to different clusters.

Must-link constraint defines transitive binary relations over instances. Consequently when we use a set of constraints, we take a transitive closure over these constraints [31]. For example, if A and B must be linked, meanwhile B and C must be linked, then A

and C must be linked certainly. Although Cannot-link constraint is not transitive, with the help of Must-link constraint, we can also extend Cannot-link constraint. For example, if A and B cannot be linked, meanwhile B and C cannot be linked, the relationship between A and C is uncertain yet. If A and B cannot be linked, meanwhile B and C must be linked, we can infer that A and C cannot be linked. Therefore, more hidden constraints can be derived using the transitivity property.

Given a test set $T = \{x_1, x_2 ... x_n\}$, in which each test x_i is represented by a feature vector of function call profile, together with the set of Must-link constraints M and the set of Cannot-link constraints C. With the limited supervised information, SSKM generates a weight matrix $W = \{w_1, w_2 ... w_d\}$ for transformation. W can transform original data into low-dimensional data $Y = \{y_1, y_2 ... y_n\}, y_i = w^T x_i$ in a more appropriate distance space and preserve the structure of original data simultaneously. The data dimension may be high for large software, such that the performance of clustering is poor. After transformation, some useless features will be filtered out and the dimension is reduced. The data instances in new distance space are more suitable for clustering than before. We use SSDR [35] to produce the transformation matrix W. To achieve this goal we establish the objective function $J(w)$ to obtain maximum value w.r.t. $w^T w=1$ [35]:

$$J(w) = \frac{1}{2n^2}\sum_{i,j}(w^T x_i - w^T x_j)^2 + \frac{\alpha}{2n_C}\sum_{(x_i,x_j)\in C}(w^T x_i - w^T x_j)^2 - \frac{\beta}{2n_M}\sum_{(x_i,x_j)\in M}(w^T x_i - w^T x_j)^2 \quad (3)$$

The objective function can be divided into three terms. The first term expresses the average squared distances of all instances. If there are few constraints, SSDR still has an acceptable effectiveness [35]. The second and third terms are the average squared distances of instances involved by pairwise constraints. In order to obtain maximum value, the objective function expands the average squared distances of instances involved by Cannot-link constraints and minus the average squared distances of instances involved by Must-link constraints.

In Equation 3, we add two parameters, α and β, to balance the contributions of constraints. If α and β are set to big values, the constraints will dominate the results. In order to leverage the significance of two constraints, we use three different pairs value of α and β. The ratio 1:1 means equal treatment for Must-link constraints and Cannot-link constraints; 1:100 means more concerned about Must-link constraints; 100:1 means more concerned about Cannot-link constraints.

Benefiting from SSDR, SSKM has an advantage that the transformed Y could be a low dimensional data set, such that clustering analysis can be performed more efficiently. Besides, the distances of instances involved by Cannot-link constraints are amplified while the distances of instances involved by Must-link are minified.

3.3 An example

In this subsection, we give a simple example to illustrate how SSKM works. We choose 4 versions of the program *schedule*, execute tests and record the failed tests for each version. In order to describe the example more concisely, six tests are selected. The detailed information of each test (x_i) with function call profile is shown in Table 1.

Table 1. Detailed Information of Tests

Tests	Function Call Profile(18 functions)
x_1	1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 1 0 1
x_2	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
x_3	0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
x_4	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
x_5	0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1
x_6	0 1 0 0 1 0 0 0 1 1 1 1 1 1 0 0 1 1

The detailed information of pairwise constraints is shown in Table 2. Four constraints are derived from test results, two are Must-link constraints: (x_5, x_6) and (x_3, x_4), and two are Cannot-link constraints: (x_1, x_2) and (x_4, x_5). The pairwise constraint (x_5, x_6) indicates that x_5 and x_6 must be in a same cluster. It is important to derive these constraints in a reasonable way for clustering by SSKM. The methods of deriving constraints from test results will be described in next section.

In order to simplify comparison, we use the value of squared Euclidean distance D. For example, $D(x_5, x_6)=6$ and $D(x_1, x_2)=D(x_3, x_4)=4$, that means the pair of tests (x_5, x_6) may be separated with higher probability by K-means than other pairs of tests. However, based on the previous test results, (x_5, x_6) and (x_3, x_4) are Must-link and (x_1, x_2) and (x_4, x_5) are Cannot-link. It is difficult to use test selection based on the cluster result because the distance space seems be not suitable for K-means. Hence a more suitable distance space should be produced with some pairwise constraints.

Table 2. Constraints Derived from Test Results

Type	Constraints
Must-link	(x_5, x_6), (x_3, x_4)
Cannot-link	(x_1, x_2), (x_4, x_5)

With the pairwise constraints in Table 2, we use SSDR [35] to generate the weight matrix W, as shown in Table 3, for transformation.

Then we can get a new data set Y as shown in Table 4. The squared Euclidean distance of each pair of tests in constraints is shown in Table 5, including both original distances and new distances after transformation. It is clear that the distances of Must-link tests are smaller than the distances of Cannot-link tests after transformation. It is more likely to get a good result of clustering as well as a good result of test selection.

Table 3. Weight Matrix *W*

W for Transformation				
0.0275	-1.0000	0.1999	-0.2702	0.0232
-0.0010	-0.0016	0.0015	-0.0001	-0.0896
-0.8432	-0.0377	0.4270	0.5201	0.1791
-0.0060	-0.0088	0.0069	0.1169	-0.2316
-0.0112	-0.0132	0.0194	0.0353	-0.9854
-0.7132	0.2414	0.3974	-1.0000	-0.0582
-0.0081	-0.0091	0.0146	0.0668	-0.4543
-0.0082	-0.0094	0.0146	0.0661	-0.4613
-0.0009	-0.0013	0.0013	0.0003	-0.0749
-0.0009	-0.0013	0.0013	0.0003	-0.0749
0.0000	0.0000	0.0000	0.0000	0.0000
-0.0010	-0.0016	0.0015	-0.0001	-0.0896
-0.0009	-0.0013	0.0013	0.0003	-0.0749
-0.0010	-0.0016	0.0015	-0.0001	-0.0896
-1.0000	-0.1365	-1.0000	-0.0603	-0.0210
-0.0062	-0.0087	0.0067	0.1176	-0.2204
-0.8432	-0.0364	0.4286	0.3854	-0.0650
-0.0113	-0.0126	0.0149	-0.0274	-1.0000

Table 4. Transformation of Tests

Tests	Transformed Data
y_1	-0.0286 -1.0594 0.2855 -1.2884 -4.8778
y_2	-3.4299 -1.0262 0.5371 -1.4332 -4.7485
y_3	-3.4566 -0.0262 0.3378 -1.1644 -4.9261
y_4	-3.4299 -1.0262 0.5371 -1.4332 -4.7485
y_5	-0.7666 0.1895 0.4824 -2.0195 -4.5652
y_6	-0.8728 -0.0741 0.4709 0.0394 -2.7098

Table 5. Squared Euclidean Distance of Tests

Constraints	Original Distance	New Distance
(x_5, x_6)	6	7.7625
(x_3, x_4)	1	1.1442
(x_1, x_2)	4	11.6709
(x_4, x_5)	4	8.9514

4. Experiment

In this section, we have conducted an experiment to evaluate the effectiveness of SSKM by comparing it with the unsupervised clustering K-means. The detailed information of experiment would be described in the following subsections.

4.1 Subject Program

We conducted the experiment with two C programs *Flex* and *Space*. The subject programs and their test sets are available from the SIR (Software-artifact Infrastructure Repository) website [36]. *Flex* is a lexical analyzer generator which receives a lexical rule and outputs a lexical analyzer. *Space* is an interpreter for an array definition language. The detailed information of them is shown in Table 6.

Table 6. Detailed Information of Subject Programs

Program Name	Number of functions	Line of code	Test suite size	Modified version
Flex	148	10459	567	5
Space	136	6199	13585	24

Cluster test selection has certain limitations when dealing with small size programs. The execution profiles of tests on small programs tend to be alike, such that it is difficult to distinguish. Therefore, the Siemens programs in SIR [36] are not adopted in our experiment.

Each subject program was provided with a correct base version and multiple single-fault versions. There are 5 modified versions for *Flex* and all of them are used in the experiment. For *Space*, we chose 24 among 38 modified versions because other versions have no or too few failed tests, which are not suitable for cluster test selection techniques [34]. These faults were injected in advance. All tests were run on each version and the test results are recorded.

4.2 Experiment Design

As depicted before, cluster analysis was used to group tests into k clusters and then the representative tests were selected according to a sampling strategy. SSKM used pairwise constraints to improve the clustering results as well as the test selection results. This limited information can be easily given by the testers or derived automatically from previous test result infomation with the constrain definitions. In the experiment, the latter method was chosen because we have already known some information in advance. And in the practice testers can also get the information easily by experience or tools.

After the tests were executed on the base version of the target program, the execution profiles would be collected. In order to obtain constraint information, we also executed tests on each modified version of each program. For each test on the modified version, we just verified whether it could detect the faults while the execution profiles would be ignored. A few of test results were used to derive different pairwise constraints by different definitions. Then K-means and SSKM were used to group tests into clusters, respectively. Finally, test selection results of K-means and SSKM were evaluated and compared to assert the effectiveness of cluster test selection. Five key steps of the experiment were designed as follows.

4.2.1 Execution Profiles Collection

Firstly we executed all tests on the base program, and then used *gcov* (a GNU tool) to collect execution coverage information, i.e. function call profile here. To collect function call profile, the entry of each function was instrumented to determine whether the function was invoked during the execution or not. On the other hand, we compared the test outputs of the base version and each modified version. The test with a different output was considered to be failed. The serial numbers of failed tests and the corresponding version numbers were also recorded. A few of them would be selected randomly to derive pairwise constraints in the next step.

4.2.2 Constraint Derivation

A main challenge of SSKM for test selection is how to generate pairwise constraints reasonably and efficiently. As mentioned before, pairwise constraints could be derived from previous test results. The process of constraint derivation would be described in this subsection.

In the application of SSKM for cluster test selection, a tester can randomly execute a few of tests. These test results are used to help constraint derivation. In order to simulate the application scenarios, we randomly generate 50 pairs of Must-link and 50 pairs of Cannot-link by cutting or repeating constraint derivation. 50 is a small number for thousands, even millions, of tests. It's easy to achieve this goal in a practical testing process.

A failed test detects at least one fault. Since there is only single fault in each modified version of program, we established mapping relations between tests and versions. $K(x)$ denotes the set of versions in which a fault can be detected by the test x. Then a coincidence degree, denoted by *CD*, is defined as follow:

$$CD(x_1,x_2)=\left|K(x_1)\cap K(x_2)\right|/\max\left\{\left|K(x_1)\right|,\left|K(x_2)\right|\right\}*100\% \quad (4)$$

where x_1 and x_2 are tests. In the definition of *CD*, $K(x_1)$ and $K(x_2)$ cannot be empty sets at the same time. That is, we do not define the coincidence degree for two passed tests.

Definition 1 (**Must-link**) Given two tests x_1 and x_2, (x_1, x_2) is Must-link if $CD(x_1,x_2) \geq TM$, in which *TM* is the threshold of Must-link. There are two definitions of Must-link with *TM*=100% and *TM*=50% in this paper.

For Must-link, the threshold *TM* could be defined as any value except for 0. *TM*= 100% is the most strict definition of Must-link. TM=50% is used to define a loose Must-link. For example, suppose $K(x_1)$ = {1,2,3,4} and $K(x_2)$ = {2,4,6}, then $K(x_1)\cap K(x_2)$ = {2,4}, (x_1, x_2) is Must-link because the coincidence degree is 50%.

Definition 2 (**Cannot-link**) Given two tests x_1 and x_2, (x_1, x_2) is Cannot-link if $CD(x_1,x_2)$=0. There are three sub-definitions of Cannot-link: (a) both x_1 and x_2 are failed tests; (b) either x_1 or x_2 is a passed test; (c) there is no special restriction for x_1 and x_2.

CD should be 0 for Cannot-link constraints. That is two Cannot-link tests should not detect any same faults. For the first sub-definition of Cannot-link, both two tests are failed ones but they do not detect a same fault. *FF* denotes this type of Cannot-link constraints. For the second sub-definition of Cannot-link, one should be a passed test and the other should be a failed test. *PF* denotes this type of Cannot-link constraints. For the third sub-definition of Cannot-link, the set of constraints is a combination of *FF* and *PF*. This type of constraints is denoted by *Any*.

It is not difficult to see that different pairwise constraints will be derived by different constraint definitions. It also will affect the results of semi-supervised clustering as well as the results of test selection. Six groups of pairwise constraints: 2 types of Must-link ×3 types of Cannot-link will be used in cluster test selection with SSKM. The comparison of these types of constraints will be discussed in Section 4.5.

4.2.3 Data Transformation

After the Must-link and Cannot-link constraints were derived, we used SSDR [35] to solve Equation 3 and generate a weight matrix *W*. Then, we used the matrix *W* to project the original data *X* on a new perspective $Y = W^T X$. In the original data, each element is a binary vector with every elements is 1 or 0 which means whether the function is called or not during the execution. After transformation, each element is a vector with floating-point values, and the size of the vector is smaller than before. It enhances the divergence degree through multiplying the original data by the weight matrix *W*.

4.2.4 Clustering

We used K-means to cluster the new data and generate a clustering result. For the purpose of comparison, we also used K-means to cluster the original data. In this step, we used the APIs of Weka [37] to extend simple K-means to SSKM. The parameter *k*, the number of clusters, was set to different values for different test set sizes. *k* was 10 for the program *Flex* because the test set is small. *k* was 50 for the program *Space* because the test set is large. In practice, the value of *k* is approximately equal to 0.5% - 2% of test set size. There is another parameter *seed* in Weka. This parameter is used to generate a random number which is, in turn, used for assigning the initial centroid of cluster. Please note that K-means is sensitive to the initial centroids. Thus, the parameter *seed* also affects the clustering results. So we repeated it 20 times with 20 different values of *seed* to complete a clustering process. The parameters of SSKM for different programs are summarized in Table 7.

Table 7. SSKM Parameters Settings

	k	Seed	α : β	Constraint Number
Flex	10	30+2* repeat	1:1,1:100,100:1	50
Space	50	50+20*repeat	1:1,1:100,100:1	50

4.2.5 Sampling

We used a popular sampling strategy, namely adaptive sampling strategy [17], to select tests. We first sampled a few of tests randomly from each cluster in 5%. If a failed test was found, then all tests in this cluster would be selected. In order to eliminate randomness, we repeated the sampling task 20 times to calculate the average value.

4.3 Evaluation

To evaluate the results of K-means and SSKM, *F-measure* was introduced to assess fault detection capability and cost reduction of regression test set comprehensively. F-measure is a combination measure of *Precision* and *Recall*, which are two widely used measures in information science.

Precision is considered as an accuracy degree of test selection, i.e. the proportion of selected failed tests in all selected tests. A high value of *Precision* indicates less waste of testing resources. Let *T'* be the set of selected tests and T'_F be the subset of failed tests in *T'*. Then *Precision* is defined as follows:

$$Precision = \frac{|T'_F|}{|T'|} \quad (5)$$

Recall is considered as a completeness measure of test selection, i.e. the proportion of selected failed tests in all failed tests. A high value of *Recall* indicates strong fault detection capability. Let T_F be the set of all failed tests and T'_F be the subset of selected failed tests. Then *Recall* is defined as follows:

$$Recall = \frac{|T'_F|}{|T_F|} \quad (6)$$

Following the definitions of *Precision* and *Recall*, both high values are expected in regression test selection. However these two measures are in conflict naturally. Improving one usually hurts another. Thus, a combination measure, *F-measure*, is introduced to evaluate the integrative benefit.

$$F-measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (7)$$

4.4 Improving SSDR Algorithm

There is a little flaw of SSDR that it employs an *n*n* matrix as a temporary variable *S* to computer the weight matrix *W*, where *n* is the size of initial test set. However, the initial test set always contains a large amount of tests, so *n* takes a big value from thousands to hundreds of thousands, even millions. Thus, the huge matrix *S* may take a lot of memory and the execution environment may run out of memory. In our experiment, we used MATLAB as a runtime environment to solve the problem of weight matrix, but when we used *Space* program, which has 13585 tests, the MATLAB is out of memory. In order to overcome the problem, we have made a slight improvement of SSDR. First, we used sparse matrix to store data. We subtracted each element of the huge matrix by a constant *a* to make most of them close to 0. When the weight matrix was produced, we added the constant *a* to each element to recover the original data. Then we used fractional step method to calculate data. Each time only a fraction of data is calculated, this make the computation time reduced sharply. The results were combined finally.

4.5 Experiment Results

The test selection results improved by SSKM are shown in Figure 2. We used box-plots to depict the improvement of SSKM to K-means, i.e., the difference of *F-measure* between two methods. As mentioned before, 5 modified versions for *Flex* and 24 modified versions for *Space* were used in this experiment. The X-axis stands for the version numbers. The Y-axis stands for the *F-measure* difference values between SSKM and K-means. In each box-plot, the band near the middle of the box is the median of *F-measure* values. A positive value indicates that SSKM outperforms K-means for test selection on average.

There two types of Must-link: 100% and 50%, three types of Cannot-link: *FF*, *PF* and *Any*. So there are six groups of constraints for SSKM. In Figure 2, we used these six groups of constraints to classify experiment results for a clear comparison.

As shown in Figure 2, we find that the *F-measure* value can be improved by SSKM in most versions for both programs. Overall, the experiment results indicate that the semi-supervised clustering method SSKM can improve the clustering results as well as the test selection results.

There is little improvement for v4 and v5 of *Flex*. We reviewed the test set of *Flex* and found that almost all tests were failed tests. The failed tests are in a dominant position for these two versions, such that the adaptive sampling strategy always selects all tests. The *Precision* and *Recall* are difficult to be improved further. Hence, we made a conjecture that we can get a better performance from the SSKM when the failed tests got a moderate proportion.

Based on the observation above, we sorted the versions of two subject programs with the rates of failed tests in ascending order in Figure 2. We find that the *F-measure* values of *Space* are promoted from v6 to v20, because of too few failed tests in the first 5 versions and too many failed tests in the last 4 versions. This observation confirms our conjecture above. The improvement of *F-measure* decreases beginning at v21. But the *F-measure* value of v24 seems an outlier. We analyzed the clustering results and found that SSKM made the clusters more pure in v24. The proportion of selected failed tests was improved, such that the *Precision* value increased and the *Recall* value remained then *F-measure* was improved.

We find that a strict definition of Must-link has a better effectiveness than a loose definition of Must-link. Taking *Flex* for example, we can obviously find that *F-measure* of *CD*=100% is prompted more significantly than *CD* = 50% for both v1 and v2. There is also some improvement for v3 when *CD* = 100%. This experiment result indicates that the noises of constraints will negatively affect test selection with SSKM. Therefore a strict constraint definition is recommended.

There is another observation on the effectiveness of different definitions of Cannot-link. It could be found in *Flex* that *PF* Cannot-link has a better effectiveness than *FF* Cannot-link. We compared the *F-measure* values of *Flex* in detail and found that the values of (100%, *PF*) are more stable than the values of (100%, *FF*). However, it has similar effectiveness for different Cannot-link definitions in *Space* program. This experiment result also indicates that the Cannot-link constraint definitions have similar effects in some applications.

As mentioned before, there is no improvement on some versions of both programs. We analyzed the results of test selection in detail and found that the *Precision* or *Recall* could also be improved in these cases. In most of these cases, the *Recall* value increased and the *Precision* value decreased. This experiment result indicates that SSKM can help us to select more failed tests, such that fault detection capability of test set is enhanced.

We summarize the above observations as follows. (1) SSKM can improve the test selection results in most versions for both programs. Although SSKM has different effectiveness on different individuals, the comprehensive results are consistent. (2) The rate of failed tests will affect cluster tests selection with SSKM. It has a better effectiveness when the failed tests are in a medium proportion. (3) The constraint definition will also affect cluster test selection with SSKM. A strict definition can usually achieve a better effectiveness in most cases.

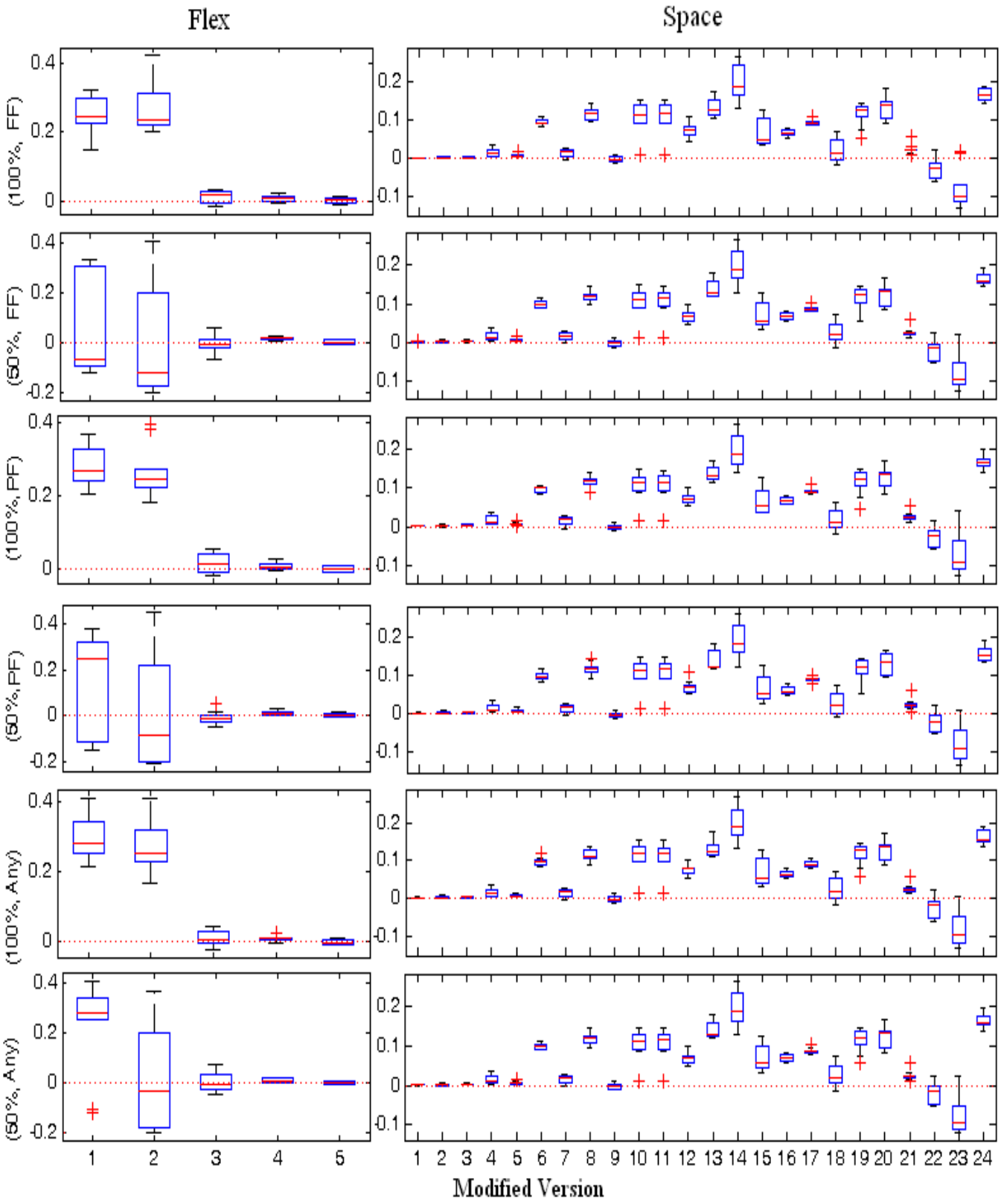

Figure 2. Improvement of SSKM (Difference of F-measure) for Flex and Space

Table 8. F-measure *t*-test results of SSKM and Simple K-means

Constraints / Programs		（100%，FF）	（50%，FF）	（100%，PF）	（50%，PF）	（100%，Any）	(50%,Any)
Flex	**h**	1	0	1	0	1	0
	p-value	0.000000232	0.564709	0.000000174	0.2690	0.000000123	0.07535
Space	**h**	1	1	1	1	1	1
	p-value	0.000002192	0.000002113	0.000001837	0.000002348	0.000001972	0.000002035

In order to compare the clustering results more formally, a standard statistical method, *t*-test is used to verify the improvement of SSKM with regard to the Simple K-means. In this experiment, we use right tailed test and set the scaling term is the value of F-measure with confidence level is 0.95. The results are shown in Table 8. We find that SSKM and improve the clustering results for Flex and Space. However we cannot draw a conclusion of improvement based on the experimental results when *CD*=50%, even SSKM may improve the clustering results actually. This is because of looser constraints definition produces many noises and the result also supports the last observation we summarize above.

4.6 Threats to validity

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 sets. In practice, the situations can be challenging. However, we believe that the subject programs are already representative of real programs in some senses, because they have been studied in many researches. On the other hand, we select the subject programs with different properties, such as different lines of code, different sizes of test sets, different data structures etc. It can also minimize the external validity in a certain degree. Threats to internal validity include the correctness of collecting execution profiles, clustering processes and constraint derivation etc. We have used a well-known profile collection program *gcov* in our previous work [32,34]. The clustering methods rely on a well-known tool Weka [37]. We use the source code [35] to implement our semi-supervised clustering method SSKM. We also try to minimize threats to internal validity by inspecting our data collection codes and results carefully.

5. Conclusions and Future Work

In this paper, we used a semi-supervised clustering method SSKM to improve regression test selection. With pairwise constraints from previous test results, the original data is transformed to a new distance space. The experiment results indicate that the semi-supervised clustering method SSKM can improve test selection in most cases. Two useful observations are made. (1) Cluster test selection with SSKM has a better effectiveness when the failed tests are in a medium proportion. (2) A strict definition of pairwise constraint can improve the effectiveness of cluster test selection.

As far as we know, this is the first paper using semi-supervised clustering for regression test selection. Many directions can be extended in the future work and two of them are: (1) SSKM is good at preserving the intrinsic structure of data as well as dimensionality reduction. Hence SSKM is not suitable for the application with a small number of features, i.e. a small program. There are many semi-supervised clustering methods and they would be studied for test selection in further. (2) As mentioned before, the clustering results depend on high quality constraints. Although we have found some observations on different definitions of Must-link and Cannot-link, it may be not sufficient in other applications. An interesting direction of our future work is to build high quality constrains for different testing scenarios.

6. ACKNOWLEDGMENTS

The authors would like to thank Professor Daoqiang Zhang. He provided the source code of semi-supervised clustering SSDR and some suggestions for this paper.

7. REFERENCES

[1] Ando, R. K. and Zhang, T. A high-performance semi-supervised learning method for text chunking. In Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics (ACL '05), 2005, pp. 1-9.

[2] Bair, E. and Tibshirani, R. Semi-supervised methods to predict patient survival from gene expression data. PLoS Biol, 2004, 2(4): e108.

[3] Black, J., Melachrinoudis, E., and Kaeli, D. Bi-criteria models for all-uses test suite reduction. In Proceedings of International Conference on Software Engineering (ICSE '04), 2004, pp. 106–115.

[4] Bowring, J. F., Rehg, J. M., and Harrold, M. J. 2004. Active learning for automatic classification of software behavior. *SIGSOFT*，2004,pp. 195-205.

[5] Belkin, M. and Niyogi, P. Semi-Supervised Learning on Riemannian Manifolds. Machine Learning, 2004, pp. 209-239.

[6] Bilenko, M., Basu, S., and Mooney, R. J. Integrating constraints and metric learning in semi-supervised clustering. In Proceedings of the 21st International Conference on Machine Learning (ICML '04), 2004, pp.11.

[7] Berkhin, P. Survey of clustering data mining techniques. Accrue Software, Inc 2002.

[8] Bradley,P. S. and Fayyad,U. M. Refining initial points for K-means clustering. Morgan Kaufmann, 1998, pp. 91-99.

[9] Basu, S., Banerjee, A., and Mooney, R. J. Semi-supervised clustering by seeding. In Proceedings of the 19th International Conference on Machine Learning (ICML'02), 2002, pp. 27-34.

[10] Basu, S., Banerjee, A., and Mooney, R. J. Active semi-supervision for pairwise constrained clustering. In Proceedings of the SIAM International Conference on Data Mining (SDM '04), 2004, pp. 333-344.

[11] Basu, S., Bilenko, M., and Mooney, R. J. A probabilistic framework for semi-supervised clustering. In Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '04), 2004, pp. 59-68.

[12] Cohn,D., Caruana,R., and McCallum,A. Semi-supervised clustering with user feedback. Technical Report, 2003.

[13] Chen, Z., Xu, B., Zhang, X., and Nie. C. A novel approach for test suite reduction based on requirement relation contraction. In Proceedings of the 23rd Annual ACM Symposium on Applied Computing (SAC'08), 2008, pp. 390–394.

[14] Demiriz, A., Bennett, K. P., and Embrechts, M. J. Semi-supervised clustering using genetic algorithms. Artificial Neural Networks in Engineering (ANNIE'99), 1999, pp. 809-814.

[15] Dai, G. and Yeung, D. Kernel selection for semi-supervised kernel machines. In Proceedings of the 24th international Conference on Machine Learning, 2007, pp. 185-192.

[16] Davidson, I. and Ravi, S. S. Clustering with constraints: feasibility issues and the k-means algorithm. In Proceedings of the 5th SIAM International Conference on Data Mining (SDM '05), 2005.

[17] Dickinson, W., Leon, D., and Podgurski, A. Finding failures by cluster analysis of execution profiles. In Proceedings of the 23rd International Conference on Software Engineering (ICSE'01), 2001, pp. 339-348.

[18] Duan, Y. W., Chen, Z. Y., Zhao, Z. H., Qian, J., and Yang, Z. J. Improving Cluster Selection Techniques of Regression Testing by Slice Filtering. In Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering (SEKE '10), pp.253-258.

[19] Engström, E., Runeson, P., and Skoglund, M. A systematic review on regression test selection techniques. Inf. Softw. Technol. 2009, 52(1):14-30.

[20] Grira, N., Crucianu, M., and Boujemaa, N. Active semi-supervised fuzzy clustering for image database categorization. In Proceedings of the 7th ACM SIGMM International Workshop on Multimedia Information Retrieval (MIR '05), pp. 9-16.

[21] Graves, T. L., Harrold, M. J., Kim, J., Porter, A., and Rothermel, G. An empirical study of regression test selection techniques. ACM Transactions on Software Engineering Methodology, 2001, 10(2):184-208.

[22] Hamerly, G. and Elkan, C. Learning the *k* in K-means. Technical Report, 2003.

[23] Harrold, M. J., Gupta, R., and Soffa, M. L. A methodology for controlling the size of a test suite. ACM Transactions on Software Engineering and Methodology, 1993, 2(3):270–285.

[24] Jain, A. K., Murty, M. N., and Flynn, P. J. Data clustering: a review. ACM Computing Surveys, 31(3): 264-323.

[25] Masri, W., Podgurski, A. and Leon, D. An Empirical Study of Test Case Filtering Techniques Based on Exercising Information Flows. IEEE Transactions on Software Engineering. 2007, 33(7): 454-477.

[26] Orso, A., Shi, N., and Harrold, M. J. Scaling regression testing to large software systems. In Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE '04), 2004, pp. 241-251.

[27] Rothermel, G. and Harrold, M. J. A safe, efficient regression test selection technique. ACM Transactions on Software Engineering and Methodology. 1997, 6(2): 173-210.

[28] Simao, A. D., Mello, R. F., and Senger, L. J. A technique to reduce the test case suites for regression testing based on a self-organizing neural network architecture. In Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC '06), 2006, pp. 93-96.

[29] Sajeev, A. S. and Wibowo, B. Regression test selection based on version changes of components. In Proceedings of the Tenth Asia-Pacific Software Engineering Conference Software Engineering Conference (APSEC'03), 2003, pp. 78.

[30] Tang, W., Xiong, H., Zhong, S., and Wu, J. Enhancing semi-supervised clustering: a feature projection perspective. In Proceedings of the 13th ACM SIGKDD international Conference on Knowledge Discovery and Data Mining (KDD '07), 2007, pp.707-716.

[31] Wagstaff, K., Cardie, C., Rogers, S., and Schrödl, S. Constrained K-means clustering with background knowledge. In Proceedings of the 18th International Conference on Machine Learning (ICML'01), 2001, pp. 577-584.

[32] Yan, S. L., Chen, Z. Y., Zhao, Z. H., Zhang, C., and Zhou, Y. M. A dynamic test cluster sampling strategy by leveraging execution spectra information. In Proceedings of Conference on Software Testing, Verification and Validation (ICST'10), 2010, pp.147-154.

[33] Yoo, S., Harman, M., Tonella, P. and Susi, A. Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge. In Proceedings of the 8th International Symposium on Software Testing and Analysis (ISSTA '09), 2009, pp. 201-212.

[34] Zhang, C, Chen, Z. Y., Zhao, Z. H., Yan, S. L., Zhang, J. Y. and Xu, B. W. An improved regression test selection technique by clustering execution profiles. In Proceedings of the 10th International Conference on Software Quality (QSIC '10), 2010, pp.171-179.

[35] Zhang, D. Q., Chen, S. C., and Zhou, Z. H. Semi-supervised dimensionality reduction. In Proceedings of the 7th SIAM International Conference on Data Mining(SDM '07), 2007, pp. 629–634.

[36] Software-artifact Infrastructure Repository. http://sir.unl.edu/, University of Nebraska

[37] Weka Home Page. http://www.cs.waikato.ac.nz/ml/weka/,University of Waikato