A Survey on Cluster Test Selection

Zhaoyi Luo, University of Waterloo

1. Introduction

Regression testing is performed when software is modified. The purpose of regression testing is to ensure the modification or newly introduced modules do not obstruct the behaviors of the existing, unchanged part of the software[1] . Ordinarily, all the existing test cases in the test suite need to be executed to ensure the integrity of the unmodified part of software, in addition to new test cases for newly added features. However, as software evolves, the test suite tends to grow that it can be unacceptably expensive to execute the entire test suite. This limitation compels researchers searching for techniques to reduce the size of test suite in regression testing.

Cluster Test Selection (CTS) aims at utilizing clustering techniques to reduce the size of test suite while the capability of the original test suite preserves. The motivation is that failures caused by the same bug is likely to have similar behaviors[2]. During the process of test selection, test cases are clustered by their “behaviors” and legacy/redundant test cases are eliminated; test cases that tend to pass are also filtered, while test cases that tend to fail remain. By selecting test cases in this way, we get a subset of test cases that imitate the fault-detection capability of the original test suite.

As a brief description of the CTS procedure, firstly test cases are abstracted into objects that are characterized by a vector of attribute values.Then a certain dissimilarity metric (e.g. n-dimensional Euclidean distance or Manhattan distance) is applied to form the distance space. Cluster analysis technique is used to cluster execution profile of test cases into clusters based on the distance space. Finally, one or more executions are sampled from each cluster or from particular clusters to form a subset of the original test suite. Existing empirical studies have shown that the application of CTS techniques is indeed cost-effective.

This paper surveys work undertaken in Cluster Test Selection area, especially those clustering techniques used to improve CTS result. In Section 2 I will introduce some background knowledge about CTS. To fully specify a cluster filtering procedure, it is necessary to select: an abstract technique to form the distance space of test cases; the particular clustering algorithm to be used, and a strategy for sampling from the clusters. So we consider each of these aspects of CTS in Section 3-5 respectively. Section 6 makes extensive discussions on issues we encountered in these research work, as well as potential future work.

2. Background

2.1. Cluster Analysis

Cluster analysis[3] is a multivariate analysis method for finding groups or clusters in a population of objects. Each object is characterized by a vector of attribute values. The goal of cluster analysis is to partition a population into clusters in such a way that objects with similar attribute values are placed in the same cluster, while objects with dissimilar attribute values are placed in different clusters. The dissimilarity of objects is measured using a distance metric, such as n-dimensional Euclidean distance.

Cluster analysis itself is not one specific algorithm, but the general task to be solved. It can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them. There are two main approaches to cluster analysis: partitioning and hierarchical clustering. The partitioning approach initially divides the population into k clusters, for a chosen k. The quality of the partitioning is improved iteratively by reassigning objects to different clusters. Although partitioning produces high-quality clusterings, it is computationally expensive. The hierarchical approach to cluster analysis is a stepwise process for generating clusters. There are two types of hierarchical clustering methods: agglomerative and divisive. Agglomerative methods initially assign each object to its own cluster. At each step a pair of clusters that is minimally dissimilar is chosen and merged to form a new cluster that replaces the chosen pair. The process continues until k clusters remain, for a chosen k. Divisive methods, on the other hand, initially assign all the objects to one cluster. At each step, a cluster is divided into two clusters. Again, the process is continued until a desired number of clusters is produced. Hierarchical methods are faster than partitioning methods, but they may not produce as good result as partitioning clustering, because it cannot reassign objects to different cluster.

2.2. Cluster Test Selection

2.2.1. Goal

In general, the goal of CTS is to utilize cluster analysis techniques to reduce the size of test suite in order to save testing cost, while the original fault-detection capability of test suite preserves. To achieve this, we need delicately design distance metrics as well as the clustering algorithms to generate good clustering result. In advance, by choosing appropriate sampling strategy on the clustering result, we can get subset of test cases that imitate the fault-detection capability of the original test suite.

As the specific goal for clustering, an ideal clustering result of test cases should satisfy the following conditions[3]: (1) test cases in the same cluster can be regarded as similar or even redundant; (2) failed test cases are separated from passed test cases; (3) two failed test cases in the same cluster are likely caused by the same fault; (4) two failures placed in different clusters because they have different causes or because they differ with respect to the nondefective code they traverse.

2.2.2. Procedure

This subsection describes the procedure of CTS[4][5]. Initially, we have a set of test cases which represents the original test suite.

Capturing Feature For this step, all test cases are executed and the execution profiles of tests will be recorded. In the profiles, each attribute is a function of program. The profiles, function call profiles provide the information of whether each is called in an execution. One way to abstract these execution profiles is to represent function call profiles as 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.

Distance Measure: For each pair of tests, the distance function is calculated. A natural way to use the Euclidean distance as the dissimilarity metric between profiles. Detailed techniques for feature capturing and distance metrics are discussed in Section 3

Cluster Analysis: Based on the distance space we built, a certain clustering algorithm (e.g. k-mean, single-linkage) is applied to partition execution profiles of test cases. The clustering result is the key base of CTS. Detailed techniques for clustering are discussed in Section 4.

Sampling: A number of strategies are available to sample from each cluster and build up a subset of test suite for regression testing. There are some commonly used strategies, such as one-per-cluster, n-per-cluster, adaptive sampling, dynamic sampling, etc. Detailed sampling strategies are discussed in Section 5.

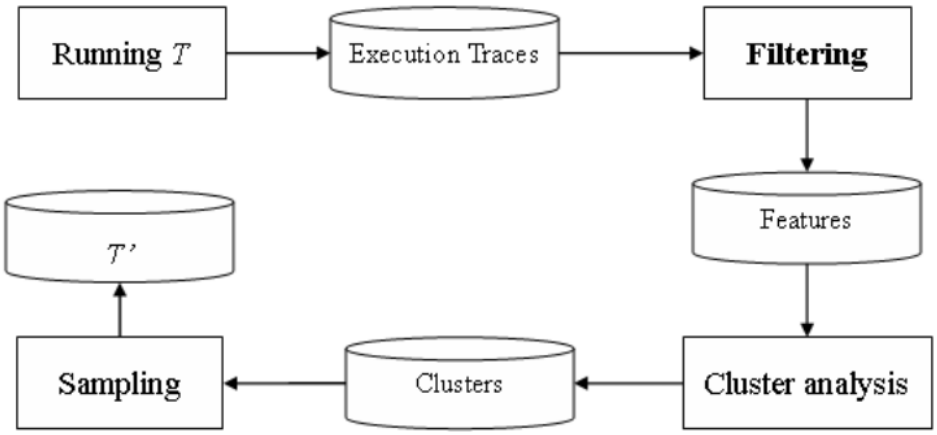

Fig.1.[6] Cluster Test Selection Procedure.

3. Feature Capturing & Distance Metrics

3.1. Overview

As the first step, we need to abstract test cases into a well-formed mathematical structure according to their execution profiles, thus to further get the distance space. By running the test, the program's executions could be represent in different level of granularity (statement, block, function, data- flow, event sequence, etc.)

The most commonly used dissimilarity metrics is n-dimensional Euclidean distance. Yet there exist several variations which apply different criteria to form the feature of profiles before applying the Euclidean distance[3]. Here we use function as the program element for demonstration.

Execution-count metric: Suppose there are *n* functions in the program under test and each function is indexed from 1 to *n.* Then each test case can be represented as: $x: \ <f_1, f_2, ..., f_n>$, where f_i is the count of invocations on the *i*th function.

Binary metric: Replace nonzero function call counts with 1 in order to emphasize differences in the coverage of program elements rather than differences in the frequency of coverage. That is to say, $x: \ <f_1, f_2, ..., f_n>$, where $f_i = 0 \ or \ 1$ .If according to the execution profile, function *i* is called then $f_i = 1$, otherwise $f_i = 0$

Proportional metric: Normalize each attribute and the range of values for each attribute is computed. Then each value is mapped to its relative position within the range.

SD metric: Normalize each attribute and the standard deviation of each attribute is computed. Then each attribute value is divided by the standard deviation.

Histogram metric: For each attribute, a histogram indicating the relative frequency of different attribute values is constructed. Each attribute value is mapped into

into 1 minus the corresponding relative frequency. This is done to emphasize attribute values that occur infrequently.
Linear regression metric: It was noticed that many attributes were correlated with the size of the program input. Linear regression technique is applied in order to eliminate that effect and expose underlying differences.
Count-binary metric: It gives equal weight to differences in program coverage and differences in the frequency of coverage.
Proportional-binary metric: It is a combination of the binary and proportional metrics. It is similar to the count-binary metric, but uses the proportion value instead of the count value.

For all the research works included in this survey, binary metric is used as their dissimilarity metric and n-dimensional Euclidean distance is applied on the binary representation. Binary metric alleviate the issue of combination explosion due to high dimensionality, and Euclidean distance is a common approach to characterize the dissimilarity of binary format of test case pairs.

3.2. Dimensionality Reduction

The properties to characterize program's executions can be statement, block, function, data-flow, event sequence, etc. However, we should notice that, it would not be a surprise to find there are even millions of statements or tens of thousand of functions in an industrial program, which induces the issue of high dimensionality.

There are several serious problems to cluster data with high dimensionality[7]: (1) The concept of distance becomes less precise as the number of dimensions grows, since the distance between any two points in a given dataset converges. The discrimination of the nearest and farthest point in particular becomes meaningless. that is to say, when dimension approximate infinity, the difference between the minimum and maximum distance reaches 0; (2) Due to the exponential growth of the number of possible values with each dimension, complete enumeration of all subspaces becomes intractable with increasing dimensionality; (3)Given a large number of attributes, some of the attributes might be meaningless for a given cluster. However the clustering result will be influenced by the "useless" attributes; (4) Given a large number of attributes, some of the attributes might be correlated. Hence, clusters might exist in arbitrarily oriented affine subspaces.

Chen et al.[6] developed a domain-independent approach to reduce high dimensionality in CTS. Program slicing technique is introduced to highlight the parts of software affected by modification of the new program version while remove the parts of software irrelevant to modification, thus to make the efficiency and effectiveness of CTS improved.

Program slicing. The process of program slicing deletes the parts of the program which have no effect upon the aspects of interest[6]. For a statement s and a set of variables {v}, the slice S of program P with respect to the slicing criterion < s, {v} > includes only those statements of P needed to capture the computing of {v} at s. In other words, the slice of program includes those statements that are directly or indirectly dependent on the values of variables in {v} at statement s. They constructed two forms of slice: backward slice and forward slice. A backward slice contains the statements of the program which can have some effect on the slicing criterion. A forward slice contains those statements of the program which are affected by the slicing criterion. The union of the backward slice and the forward slice is the final slice S. S is the dimensionality-reduced version of test case features that magnify the difference between test cases regarding with the modification of program, and ignore the difference beyond the modification. In this way, two test cases quite different can be regarded as the same with respect to program slice due to their similarity on the modified part of program (t2 and t3 in Fig.2 is such an example). Empirical study shows program slicing technique indeed helps generate better clustering result, and run faster.

No.	Statement	Slice	t_1 (1,6)	t_2 (2,0)	t_3 (7,6)
	viod f (int m, int n){				
1	int a=0, b=0, c=0, d=0;	⋆	●	●	●
2	if (m<n)	⋆	●	●	●
3	a=1;	⋆	●		
	else				
4	b=1;			○	○
5	while(a>0){ /* (a>=0)*/	⋆	●	●	●
6	c=c+a;	⋆	●		
7	a=a−1;}	⋆	●		
8	printf("%s",c);	⋆	●	●	●
9	if(n>0){		○	○	○
10	b=n−5;		○		○
11	while(b>0){		○		○
12	d=d+b;		○		○
13	b=b−1;}		○		○
14	printf("%s",d);}		○		○
	}				

Note: The modification is in the statement s_5. ⋆ denotes the statement in the program slice. ● denotes the statement remained by slice filtering. ○ denotes the statement removed by slice filtering.

Fig.2.[6] A Sample of Program Slicing.

Dynamic Distance Metric

To accurately capture the dissimilarity of test cases, Wang et al.[4] developed an approach to dynamically build the distance space -- test cases are characterized as binary strings while each dimension represent whether the test case calls a certain function. With feedback information from execution of test cases, functions are ranked in a dynamic way according to the possibility whether the function might contain fault. Distance space is adjust in such a way that difference in dimensions which represent high-ranking functions are magnified while difference in dimensions which represent low-ranking functions are minified. Thus each attribute is given a weight according to its "importance" to clustering.

Weighted Attribute based Strategy (WAS). WAS calculate the likelihood of every test case to fail iteratively. The test case with highest ranking to fail is regarded as "suspicious" and it will be executed. If the execution fails then all the functions it invokes decrease their "confidence", otherwise all the functions it invokes increase their confidence. The function with the least confidence is regarded as suspicious function. All the test cases that invokes this function are selected as suspicious test cases which are executed again. By such an iterative process, we get the ranking of all functions that can be used as the weight of attributes.

Here is a brief description of the procedure of WAS：

1. Test selection. Select a test which has the highest suspiciousness.
2. Function confidence calculation. Selected function is executed. The confidence of each functions it invokes increase by 1 if it passes, otherwise decrease by 1if it fails.
3. Function selection. Select functions that have confidence less than a certain threshold as suspicious functions. Then all suspicious functions are used to compute the suspiciousness of unselected test cases -- if a test case invokes a suspicious function, then its suspiciousness increases. The suspiciousness is used in step 1.

The above procedures iterate. Finally the function ranking information will be used to calculate the weight of each attribute and adjust the distance measurement in order to get better clustering result. However, in my opinion the high dimensionality issue still exist for this approach. If the dimension of attribute is very high, the effect of such adjustment makes little influence on clustering result.

4. Clustering algorithms

4.1. Overview

Clustering algorithm is the core part of CTS. Given the distance metrics of test cases, clustering algorithm directly affect the final result. Given dissimilarity function, the goal of clustering is to group test cases in such a way that test cases in the same group are more similar to each other than to those in other groups. There are mainly two kinds of clustering algorithms used in CTS: agglomerative hierarchical algorithms and partitioning algorithms. Dickinson et al.[3] justify that agglomerative hierarchical clustering is used in their CTS experiments because it is faster than partitioning and because it performed reasonably well in preliminary experiments. Besides, Chen et al. justify that k-mean (a kind of partitioning algorithm) is used in all their experiment because it is widely used and it generates good result.

Unfortunately, Research works seldom target at improving clustering algorithms to improves the clustering result of CTS. In my opinion, there are mainly three reasons for this phenomenon:

1) General clustering task is based on given, reliable distance metrics. However distance in CTS is not reliable. In CTS it is assumed that test cases have similar behavior are likely hit the same fault. Based on this assumption we build distance metrics to reflect the behavior of test cases as well as validate our distance metrics. If clustering result is not good, people may naturally criticize the distance metrics does not reflect the actual software behavior.
2) The clustering result in CTS can not be judged directly (e.g. by manually inspection). It is not possible to directly evaluate the clustering result of test cases until we take samples from each cluster, form the subset of test suite and run it to measure its fault-detection capability. So sampling strategy need to be designed, and the feasibility of sampling strategy also need to to be verified.
3) Reliable distance metric is designed to make sure it reflects actual software behavior, and reliable sampling strategy is designed to make sure fault-revealing test cases are selected. Therefore, domain-based knowledge are used to improve distance metrics and sampling strategy. In contrast, clustering algorithm is domain-independent. Improving clustering algorithm requires domain-independent characterization of data under cluster, which again, is based on reliable distance metrics.

4.2. Semi-supervised Clustering

Chen et al.[5] introduced a semi-supervised clustering method -- semi-supervised K-means (SSKM), to improve cluster test selection. SSKM utilizes limited supervision information derived from previous test results to guide the clustering process. Labeled data is often useful for the learning process, but labeled data is often limited and expensive to generate, since the labeled information typically requires human expertise and domain knowledge to get. In CTS, we can derive appropriate information from previous test results in the form of pairwise constraints: Must-link and Cannot-link. Must-link means two test cases must be in the same cluster, and Cannot-link means two test cases must not be in the same cluster. Thus both unlabeled and labeled data are available in the application for clustering.

In SSKM, Must-link constraint defines transitive binary relations over instances, so a transitive closure can be deducted over these constraints. Although Cannot-link constraint is not transitive, with the help of Must-link constraint, more constraint can also be extended from Cannot-link. For example, If A and B cannot be linked, while B and C must be linked, we can infer that A and C cannot be linked. Therefore, more constraints can be derived from the original set of constraints.

Next, SSKM uses the semi-supervised information to change the distance metric in this way: The distance between two test cases with a Must-link constraint is largely shortened so that they are likely to be put in the same cluster during the clustering process; similarly, the distance between two test cases with a Cannot-link constraint is largely lengthened so that they are unlikely to be put in the same cluster. More specifically, 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 while the structure of original data preserves. After transformation, some useless features will be filtered out and the dimension is reduced. The objective function $J(w)$ is established to obtain maximum value
with respect to $w^T w = 1$, in order to produce the transformation matrix W by SSDR[8]. For any two test cases i and j, and set of Must-link constraints M and set of Cannot-link constraints C:

$$J(w) = \frac{1}{2n^2}\sum_{i,j}(w^T x_i - w^T x_j)^2$$

$$+ \frac{\alpha}{2n_C}\sum_{(x_i,x_j)\in C}(w^T x_i - w^T x_j)^2$$

$$- \frac{\beta}{2n_M}\sum_{(x_i,x_j)\in M}(w^T x_i - w^T x_j)^2$$

The objective function can be divided into three parts. The first part is the normal Euclidean distance between any two test cases. The second part applies for any two test cases with Cannot-link. α is a big value which means their distance will be largely lengthened so that they are unlikely to be put in the same cluster. The third part applies for any two test cases with Must-link. β is a also big value which means their distance will be largely shortened so that they are likely to be put in the same cluster. Finally, k-mean clustering algorithm is applied on the adjusted distance space.

5. Sampling Strategies

5.1. Overview

After the cluster analysis step, a subset of the test cases is chosen for evaluation. There are several sampling strategies:

Random sampling This is most basic sampling strategy, which involves selecting test cases from the entire test suite randomly, and each sample has equal probability of being selected. This procedure is often used as the baseline against other more advanced sampling strategy..

One-per-cluster (or n-per-cluster) sampling Selects one (or n) member at random from each cluster. With this strategy, more executions are selected from small clusters than from large ones, because small clusters are typically much more numerous. It is supposed a significant proportion of failures are isolated in small clusters, so this strategy should be effective at finding them[3].

Adaptive sampling Initially, one test case is selected randomly from each cluster. All selected test cases are executed. If a selected test case fails, then all other test cases of its cluster are also selected. Like one-per-cluster sampling, this strategy favors executions in small clusters. Research shows its more efficient than n-per-cluster strategy.

Execution-spectra-based sampling (ESBS) ESBS is a form of adaptive sampling. Different from the normal adaptive sampling strategies, ESBS iteratively selects test cases from each cluster. Selection is based on suspiciousness of test cases which is computed dynamically on the execution spectra information of previous selected test cases. Detailed procedure of this sampling strategy is described in the following subsection.

5.2. Execution-spectra-based Sampling

Execution-spectra-based sampling[9] (ESBS) computes suspiciousness of test cases dynamically, based on the execution spectra information of previous execution of selected test cases. Specifically, ESBS computes a suspiciousness value for each test case. If the suspiciousness value is larger than a predefined threshold, then the corresponding test is considered to be “suspicious”. Next ESBS uses the inspection result (i.e. the actual pass or fail information) and the execution spectra information of the selected suspicious test to update the suspiciousness of the remaining tests. This selection process is repeated until there is no suspicious test available. The experimental results show that ESBS is more effective than most existing test cluster sampling strategies[9].

we should notice ESBS is a sampling strategy from the entire population, which means ESBS is independent from clustering techniques. To adapt it to the clustering result, we can apply ESBS for each cluster separately. ESBS is also applicable on different granularity of test case featuring (function, block, statement, etc.). Here statement-level granularity is used for illustration.

ESBS is composed of five phases:

1. Test selection. select a test case with the highest suspiciousness from the population. If there are more than one with the highest suspiciousness (e.g. the initial situation), randomly select one from them.
2. Result inspection. assign the selected test case to testers for result inspection. The execution spectrum (passed or failed) will influence the set of suspicious statements and further influence the set of suspicious tests.
3. Statement measurement: re-compute the confidence of each statement based on the execution spectrum of selected test case. If it is passed, decrease the confidences of the statements it executed. Otherwise, If it is failed then decrease the confidences of the statements it executed.
4. Statement identification: statements are identified to be correct or suspicious based on their confidence value with a given confidence threshold.
5. Test measurement: update the suspiciousness of the remaining test cases.The suspiciousness of a test case equal to the number of suspicious statements it executes.

Step 1-5 iterates until no suspicious test cases left in the population. All the suspicious test cases we selected form a subset of the original test suite, which is the final result of ESBS.

This approach resembles Weighted Attribute based Strategy (WAS) introduced in Section 3.3. WAS takes advantage of ranking of functions, and change the weight of attribute to form new distance space. Thus influence of attributes corresponding to suspicious functions is magnified. In contrast, ESBS takes advantage of ranking of test cases, and select test cases with high suspiciousness as the sample.

6. Discussion

In this section, I summarize my opinions and critiques according to each topic, especially those closely related to clustering theory.

6.1. Dimensionality Reduction

High dimensionality is a key issue for clustering. There are serious problems to cluster data with high dimensionality[7]: (1) The concept of distance becomes less precise as the number of dimensions grows, since the distance between any two points in a given dataset converges. The discrimination of the nearest and farthest point in particular becomes meaningless. that is to say, when dimension approximate infinity, the difference between the minimum and maximum distance reaches 0; (2) Due to the exponential growth of the number of possible values with each dimension, complete enumeration of all subspaces becomes intractable with increasing dimensionality; (3)Given a large number of attributes, some of the attributes might be meaningless for a given cluster. However the clustering result will be influenced by the “useless” attributes; (4) Given a large number of attributes, some of the attributes might be correlated. Hence, clusters might exist in arbitrarily oriented affine subspaces.

In CTS, the dimension of data depends on the granularity of test case featuring (e.g. function-level, statement-level), and is usually very high. It is not a surprise there are millions of statements and tens of thousand functions in a medium-size industrial program. Thus for CTS, dimensionality reduction is a particular important task.

For all the research work surveyed, binary metric is chosen as the test case featuring method. This is partially because choosing binary-value for each attribute helps to alleviate the exponential growth of the number of possible values with each dimension. Many of these work choose function-level rather than statement-level of granularity also help alleviate the high dimensionality problem.

SSDR[8] introduces a semi-supervised dimensionality reduction approach. In this case, besides abundant unlabeled examples, domain knowledge in the form of pairwise constraints are available, which specifies whether a pair of instances belong to the same class (must-link constraints) or different classes (cannot-link constraints). The dimension can be reduced while preserving the intrinsic structure of the unlabeled data as well as both the must-link and cannot-link constraints defined on the labeled examples in the projected low-dimensional space. SEMI applied SSDR in CTS as the semi-supervised dimensionality reduction method.

For other research work where high dimensionality problem is not concerned, I suppose the dimensionality problem still exist which affect the efficiency and effectiveness of CTS in large.

6.2. Clustering Algorithms in CTS

It is an interesting phenomenon that research works seldom target at optimizing clustering algorithm to improve clustering result in CTS. As we know, general clustering task is based on given, reliable distance metrics. However distance in CTS is not reliable. In CTS it is assumed that test cases have similar behavior are likely hit the same fault. Based on this assumption we need to use domain knowledge to build distance metrics to reflect the behavior of test cases, which takes higher priority than optimizing domain-independent clustering algorithm. On the other hand, the clustering result in CTS can not be judged directly unless we take samples from each cluster, form the subset of test suite and run it to measure its fault-detection capability. So sampling strategy need to be designed, and the feasibility of sampling strategy need to to be verified, which also takes high priority. It can be interesting to study how different clustering algorithms influence the clustering result.

6.3. Featuring & Data Characterization

Reliable test case featuring method and distance metric should be designed to make sure it reflects actual software behavior. Many aspects of a program's executions may be profiled in order to characterize its behavior. Such aspects include control flow, data flow, variable values, and event sequences[10]. Thus domain-based knowledge need to be heavily used to improve distance metrics. So far, research works often use binary representation on test cases with various level of granularity (e.g. function, block, statement), and use Euclidean distance as the distance metric. I suppose we can try to explore more ways to characterize test case (e.g. use other distance metric rather than Euclidean distance).

So far, no research work aims at studying the pattern of data under test in CTS. For example, after transforming test cases into binary strings, we can learn how they distribute in their Euclidean space. It is possible that after we know more about the data under cluster, more clues may rise to help optimize the clustering algorithm.

Reference:

[1] Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, *7*, 1–7. doi:10.1002/000

[2] Chao Liu, Xiangyu Zhang, and Jiawei Han. A systematic study of failure proximity. *IEEE Trans. Software Eng.*

[3] Dickinson, W., Leon, D., & Podgurski, A. (2001). Finding failures by cluster analysis of execution profiles. *Proceedings of the 23rd …*, 339–348. Retrieved from http://dl.acm.org/citation.cfm?id=381509

[4] Wang, Y., Chen, Z., Feng, Y., Luo, B., & Yang, Y. (2012). Using Weighted Attributes to Improve Cluster Test Selection. *2012 IEEE Sixth International Conference on Software Security and Reliability*, 138–146. doi:10.1109/SERE.2012.18

[5] Chen, S., Chen, Z., Zhao, Z., Xu, B., & Feng, Y. (2011). Using semi-supervised clustering to improve regression test selection techniques. *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, 1–10. doi:10.1109/ICST.2011.38

[6] Engineering, K., Chen, Z., Duan, Y., Zhao, Z., Xu, B., Qian, J., & Uni-, N. (2011). Using Program Slicing to Improve the Efficiency and Effectiveness of Cluster Test Selection.

[7] Kriegel, Hans-Peter; Kröger, Peer; Zimek, Arthur (2009), "Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering", ACM Transactions on Knowledge Discovery from Data (New York, NY: ACM) 3 (1): 1–58, doi:10.1145/1497577.1497578

[8] 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.

[9] Yan, S., Chen, Z., Zhao, Z., Zhang, C., & Zhou, Y. (2010). A Dynamic Test Cluster Sampling Strategy by Leveraging Execution Spectra Information. 2010 Third International Conference on Software Testing, Verification and Validation, 147–154. doi:10.1109/ICST.2010.47

[10] Dickinson, W., Leon, D., & Podgurski, A. (2001). Pursuing Failure : The Distribution of Program Failures in a Profile Space, 246–255.