CSIT (September 2013) 1(3):253–260
DOI 10.1007/s40012-013-0023-3

ORIGINAL RESEARCH

Software testing optimization through test suite reduction using fuzzy clustering

Gaurav Kumar · Pradeep Kumar Bhatia

Received: 22 August 2012 / Accepted: 23 July 2013 / Published online: 13 August 2013

Abstract Software testing is a most important but expensive activity. To get the most efficient and effective testing, test cases are designed on the basis of conditions. While designing test cases, many test cases are developed that are of no use or produced in duplicate. Exhaustive testing requires program execution with all possible combinations of values for program variables, which is impractical due to resource limitations. Redundant test cases or the test cases that are of no use, simply increases the testing effort and hence increases the cost. Our goal is to reduce the time spent in testing by reducing the number of test cases. For this we have incorporated fuzzy techniques to reduce the number of test cases so that more efficient and accurate results may be achieved. Fuzzy clustering is a class of algorithms for cluster analysis in which the allocation of similar test cases is done to clusters that would help in finding out redundancy incorporated by test cases. We proposed a methodology based on fuzzy clustering by which we can significantly reduce the test suite. The final test suite resulted from methodology will yield good results for conditions/path coverage.

Keywords Cyclomatic complexity · Equivalence class · Fuzzy c-means clustering · Standard deviation · Test case redundancy

Electronic supplementary material The online version of this article (doi:10.1007/s40012-013-0023-3) contains supplementary material, which is available to authorized users.

G. Kumar (✉) · P. K. Bhatia
Department of Computer Science & Engineering, Guru Jambheshwar University of Science & Technology, Hisar, Haryana, India
e-mail: er.gkgupta@gmail.com

P. K. Bhatia
e-mail: pkbhatia.gju@gmail.com

1 Introduction

Software testing is one of the important processes of software engineering discipline. Software testing process is time consuming and costly, so the size of test suite plays an important role. A test suite is developed for the initial version of the software and reused to test each successive version of the software. A test case is defined in IEEE standard as: "A set of test inputs, execution, and expected results developed for a particular objective such as to exercise a particular program path or to verify compliance with a specific requirement". A test suite is a combination of test cases. The quality of a test suite can be measured through following four factors: (1) its fault coverage, (2) its code coverage, (3) its size, and (4) the number of faults detected by the most effective test contained in it. To test new or changed requirements, new test cases are developed and added to the test suite. So the size of test suite grows and the cost of running it on the modified software (i.e. regression testing) increases. Therefore, the idea of test-suite reduction is to find a minimal subset of the test-suite that is sufficient to achieve the given test requirements or in other terms the conditions to satisfy the requirements of the program. Existing test-suite reduction techniques [1] consider test-case coverage criteria (e.g. statements, decisions, definition-use associations, or specification items). Reduction of the size of test-suites [2] is necessary because continuous growth of test cases may affect the cost of maintenance. Test-suite reduction can be achieved by removing the redundant test cases. Test cases should be reduced in a manner so that, neither test coverage nor fault detection ability is degraded [3–5]. Many techniques have been developed for reducing number of test cases. The quality of services [6] is a key issue for developing service-based software systems and testing is necessary for

evaluating the functional correctness, performance and reliability of individual as well as composite services. Data mining [7] in which clustering techniques are used and in which clusters similar data that can be applied to the test suite to significantly reduce the test suite. Many techniques for test case reduction are available like equivalence class testing, boundary value testing, pair wise testing etc [8–10].

2 Background

In this paper, we propose a method which removes the test case redundancy by use of fuzzy clustering technique. Using fuzziness while doing clustering provides more accurate results. Before going to the proposed methodology, in the next few sub sections we shall learn some basic understanding of concepts related to software testing and fuzzy clustering. The proposed technique is very similar to equivalence class testing. In equivalence class method [11], input domain of a program is partitioned into a finite number of equivalence classes such that one can reasonably assume, but not be absolutely sure, that the test of a representative value of each class is equivalent to a test of any other value. In our methodology, instead of classes we are using fuzzy clusters that may provide more accurate and efficient results.

2.1 Software testing

Testing leads to uncovering problems which enhances further debugging. Software deployed without testing leads to unreliability. Hence testing the software to the full extend is a necessary task while building a software. Testing the software implies executing possible test cases. The extent of testing can be evaluated using several techniques like path coverage, conditional coverage, code coverage etc. This phase of Software Development Life Cycle (Khan et al. [12]) is the most expensive phase. It requires lot of time and effort. Hence optimization of test cases is a must. But first we need to see what a test case looks like.

A test case [7] is a collection of different inputs for the software. So a test case can be compared to a tuple in a database table. It has ID, $input_1$, $input_2$,…, $input_n$. A good test case is one that is able to find faults with the software. Hence the output of test case will be a pass/fail. Length of test case also has an impact on resulting test suites. A longer test case [13] finds more difficult faults but reduces the number of test cases that are necessary to achieve the test objectives. Also, a longer test case has disadvantages such as higher computational costs and is more difficult to interpret manually.

A test suite TS is a finite set of n test cases. The overall length of test-suite TS is the sum of the lengths of its test-cases t_i: $length\ (TS) = \sum_{i=1}^{n} length\ (t_i)$. Redundancy may be started in the process of test data generation because of the process of automation. Redundancy is the repetition of data between one test case and the other. So to save lot of time from executing redundant or unnecessary test cases, optimization of test suite is important to achieve. The behavioural patterns exhibited by the test suite helps us in this process of automation [14, 15].

Cyclomatic complexity [16, 17] is the number of different paths that can be followed to get from the beginning to the end of a method. To adequately test a function having lot of branches and possible paths is more complex and more difficult and is more prone to having undiscovered bugs as compared to a simple method with only a few different paths through the code [18]. In spite of all this, a little attention has been paid to cyclomatic complexity in the software development industry. This may be due to development teams not having adequate tools to quickly and easily measure the cyclomatic complexity of their code. The difficulty of adequately testing methods rises quickly as the complexity goes up. The reason for using cyclomatic complexity as an initial guess for number of clusters in our proposed methodology is that we require some reliable initial point at which we can start our fuzzy clustering mechanism. For selecting initial number of clusters, cyclomatic complexity may be omitted but then it would take more effort as every time centroid of the clusters has to be computed and on the basis of that increase/decrease number of clusters.

2.2 Fuzzy clustering

In hard clustering, data is divided into distinct clusters, where each data element belongs to exactly one cluster. In fuzzy clustering (also referred to as soft clustering), data elements can belong to more than one cluster, and associated with each element is a set of membership levels. These indicate the strength of the association between that data element and a particular cluster. Fuzzy clustering is a process of assigning these membership levels, and then using them to assign data elements to one or more clusters. *Fuzzy clustering* [19, 20] allows each feature vector to belong to more than one cluster with different membership degrees (between 0 and 1) and vague or fuzzy boundaries between clusters. In fuzzy clustering, each point has a degree of belonging to clusters as in fuzzy logic, rather than belonging completely to one cluster only. Thus, points on the edge of a cluster may be in the cluster to a lesser degree than points in the center of cluster. Any point x has

a set of coefficients giving the degree of being in the k_{th} cluster $w_k(x)$ [21].

2.3 Difficulties with fuzzy clustering

- The optimal number of clusters C to be created has to be determined so a good cluster validity criterion has to be found.
- The character and location of cluster centroid is not necessarily known a priori, and initial guesses have to be made.
- The data characterized by large variability in cluster shape, cluster density, and the number of points (feature vectors) in different clusters has to be handled.

2.4 Fuzzy c-means clustering algorithm

One of the most widely used fuzzy clustering algorithms, which allow one piece of data to belong to two or more clusters, is the *Fuzzy C-Means* (*FCM*) *Algorithm*. The FCM algorithm [22–25] attempts to partition a finite collection of n elements $X = \{x_1, \ldots, x_n\}$ into a collection of c fuzzy clusters with respect to some given criterion. Given a finite set of data, the algorithm returns a list of c cluster centers $C = \{c_1, \ldots, c_c\}$ and a partition matrix

$$M = u_{i,j} \in [0,1], \quad i = 1, \ldots, n; \; j = 1, \ldots, c$$

where each element $u_{i,j}$ tells the degree to which element x_i belongs to cluster c_j. The objective function which is to minimize is:

$$J_m = \sum_{i=1}^{N} \sum_{j=1}^{C} u_{ij}^{m} \left\| x_i - c_j \right\|^2, \quad 1 \le m < \infty \tag{1}$$

Here the membership values u_{ij} and the fuzzifier m is added. The fuzzifier m determines the level of cluster fuzziness. A large m results in smaller membership u_{ij} and hence, fuzzy clusters results. In the limit $m = 1$, the memberships u_{ij} converge to 0 or 1, which implies a crisp partitioning. In the absence of experimentation or domain knowledge, m is commonly set to 2. $\|*\|$ is any norm expressing the similarity between any measured data and the center. The standard function is:

$$u_{ij} = \frac{1}{\sum_{k=1}^{C} \left(\frac{\left\| x_i - c_j \right\|}{\left\| x_i - c_k \right\|} \right)^{\frac{2}{m-1}}} \tag{2}$$

With fuzzy c-means, the centroid of a cluster is the mean of all points, weighted by their degree of belonging to the cluster:

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^{m} \cdot x_i}{\sum_{i=1}^{N} u_{ij}^{m}} \tag{3}$$

The degree of belonging u_{ij}^{m} is related inversely to the distance from x to the cluster center. It also depends on a parameter m that controls how much weight is given to the closest center. The steps followed in fuzzy c-means algorithm are:

- *Choose a number of clusters* There are two main approaches to determine the appropriate number of clusters. One is *Validity Measures* that are scalar indices that assess the goodness of the obtained partition. Clustering algorithms generally aim at locating well separated and compact clusters. When the number of clusters is chosen equal to the number of groups that actually exist in the data, it can be expected that the clustering algorithm will identify them correct. Second approach is *Iterative merging or insertion of clusters*. The basic idea of cluster merging is to start with a sufficiently large number of clusters, and successively reduce this number by merging clusters that are similar (compatible) with respect to some well defined criteria.
  For our proposed algorithm, we are using first one i.e. *Validity Measures* that will represent cyclomatic complexity drawn through independent paths of the graph.
- Assign randomly to each point coefficients in the clusters.
- Repeat until the algorithm has converged (the coefficients change between two iterations is no more than ε i.e. $\max_{ij}\left\{ \left| u_{ij}^{(k+1)} - u_{ij}^{(k)} \right| \right\} < \varepsilon$, where ε is the given sensitivity threshold i.e. a termination criterion that lies between 0 and 1; and k are the iteration steps):
  - Compute the centroid for each cluster, using the above formula.
  - For each number of clusters, calculate the aggregate of centre of clusters and then calculate standard deviation from them. Wherever there is minimum standard deviation found, that will be the optimum number of clusters we require for the problem.

2.5 Fuzzy logic toolbox

Fuzzy Logic Toolbox command line function fcm starts with an initial guess for the cluster centers, which are intended to mark the mean location of each cluster. The initial guess for these cluster centers is most likely incorrect. Additionally, fcm assigns every data point, a membership grade for each cluster. By iteratively updating the cluster centers and the membership grades for each data point, fcm iteratively moves the cluster centers to the right location within a data set. This iteration is based on minimizing an objective function that represents the distance

from any given data point to a cluster center weighted by that data point's membership grade.

The command line function fcm outputs a list of cluster center and several membership grades for each data point. Information returned by fcm can be used to help in building a fuzzy inference system by creating membership functions to represent the fuzzy qualities of each cluster.

2.6 Select cluster algorithm

C-means algorithm [7] clusters test cases on the basis that test cases in the same cluster have the same behaviour. If we test different test cases from the same cluster, this would add redundancy because they would exhibit the same results. So in order to reduce this redundancy we need a selective approach of choosing test cases. The step given below proposes a method to choose tuple randomly from a cluster.

Step1: *Input* Clustered data points (all test cases) from c-means clustering algorithm.

Step2: *Output* Single data point i.e. centroid (reduced test case) from each Cluster.

Step3: *Process* to follow:

- For each cluster (1,…, i), where each C_i contains all the tuples (t_{i1},…, t_{in}) that are mapped to it from c-means.
- Select one tuple randomly from (t_{i1}, t_{i2},…, t_{in}).
- Add selected tuple with the label C_i to output file.

At the end of execution of this algorithm we have one tuple from each cluster.

3 Proposed methodology

Based on the idea of software testing and fuzzy clustering, the methodology proposed in this section deals with the efficient reduction of test suite. Proposed algorithm in stepwise manner is given as under:

(1) Generate test cases for the given problem either manually or using automated tool.
(2) Determine cyclomatic complexity to initially assign the number of clusters.
(3) Apply fuzzy c-means clustering algorithm to the above generated data. Here c signifies how many clusters or in other words how many test cases we are looking for. The initial value of c will be equal to the cyclomatic complexity.
(4) Apply select cluster algorithm on the clusters formed through applying c-means clustering algorithm. Execute the algorithm and save the output in a file.
(5) Use this saved file to check for coverage of condition/path for the software. This can be determined by using the Table 1. Multiple conditions may lie under a single cluster.
(6) Until possible coverage for the conditions is achieved, assign value of c to somewhat greater than previously defined and repeat from step3. This can be achieved by calculating standard deviation of aggregate of cluster centroids. The point at which the standard deviation stops declining further and starts increasing will define the optimum number of clusters that will provide good coverage for the problem with reduced redundancy.

Effectiveness of the test suite minimization [26] can be calculated as follows:

$$\left(1 - \frac{\text{Number of test cases in the reduced test suite}}{\text{Number of test cases in the original test suite}}\right) \times 100\ \%$$

The impact of test suite minimization can be measured as follows:

$$\left(1 - \frac{\text{Number of faults detected by the reduced test suite}}{\text{Number of faults detected by the original test suite}}\right) \times 100\ \%$$

4 Example of proposed algorithm

Consider the problem for the determination of the nature of roots of a quadratic equation. Its input is a triple of positive integers (say a, b, c) and value may be from interval [0,100]. The output may have one of the following words:

[Not a quadratic equation; real roots; imaginary roots; equal roots].

Here for the above example, we are using automated test case generation and from that five hundred test cases are considered. Cyclomatic complexity for the given problem describes that the test cases exhibits seven different behaviours. *Validity Measures* for initial value of Clusters that will represent cyclomatic complexity to achieve a good coverage of the test cases trying to achieve highest test case reduction. A limited number of test cases each having different behaviour may be sufficient enough to test for these seven different paths. So applying the methodology proposed in the previous section by clustering these test cases. Let us choose initial dummy 'c' value for the

Table 1 Condition and corresponding cluster defining coverage of the conditions

Conditions	Cluster
Cond. 1, 2, 3,…, j	C_1
Cond. 2, 3,…, j	C_2
Cond. i, i + 1,…, j	Cc

Table 2 Initial conceptual condition and corresponding cluster based on cyclomatic complexity

Conditional statements	Cluster, C_i
If((a >= 0) && (a <= 100) && (b >= 0) && (b <= 100) && (c >= 0) && (c <= 100)) //results in the values do not constitute a quadratic equation	C_1
If((a >= 0) && (a <= 100) && (b >= 0) && (b <= 100) && (c >= 0) && (c <= 100)) //results in the inputs belong to invalid range	C_2
If!(a = 0) //results in the inputs belong to invalid range	C_3
If(validInput = −1) //results in the values do not constitute a quadratic equation	C_4
If(validInput == 1) && if(d == 0) //results in the roots are equal	C_5
If(d > 0) //results in the roots are real	C_6
If(d < 0) //results in the roots are imaginary	C_7

algorithm as seven and then clustering these test cases into clusters of different behaviour.

So the output of the above algorithm in iteration one gives seven different test cases. Test the above algorithm for coverage as per the Table 2. If all the paths/conditions are not covered then repeat the process by taking some higher value of c and until good coverage is achieved. By this approach we reduced the time wasted in testing unnecessary test cases. Further, effectiveness of the test suite minimization and impact of test suite minimization can be measured by the procedure given in proposed methodology section.

The methodology is applied by taking a dummy example of three variables where it reduces the test suite significantly. The technique may be further tested on programs that contain many variables, several conditional checks and coverage of the conditions for different values of c. As the c value increases the number of test cases to be tested increases and also there is improvement in coverage.

5 Evaluation and result analysis

On the basis of cyclomatic complexity, when we take initial number of clusters as seven for the given problem we found the clusters and convergence as shown in Fig. 1a through MatLab v2010. As seven is the initial guess for the number of clusters, it has to be checked by increasing number of clusters for more minimized objective function with respect to Iteration Count so that deviation in center of cluster is minimum.

Increasing the number of clusters 8, 9, 10, 11 as shown in Fig. 1(b–e), we got lesser value of objective function. Increasing number of clusters, objective function value gets minimized i.e. there is improvement in coverage. As we have to reduce number of test cases with good coverage, so we have to balance objective function with number of clusters. So, while applying fuzzy c-means clustering algorithm, centroid of each clusters are also calculated for the number of variables (i.e. three for our dummy example). On the basis of aggregate of centroid of each cluster, we calculate the standard deviation as shown in Table 3 and drawing a graph for the same as shown in Fig. 2. The average running time of the testing will be: test case generation + cyclomatic complexity calculation + cluster generation + select cluster + testing of the software with the new test suite.

As shown in Table 3 and Fig. 2, when the numbers of clusters are nine, deviation among cluster centres' is minimum as compared to deviation in other cluster centres or in other words number of reduced test cases for the given problem. So nine will be the minimum no. Of reduced test cases for the given problem will overall coverage.

6 Limitations and future work

Here for the proposed methodology, initial guess of number of clusters has to be made. In our proposed methodology we are taking it as cyclomatic complexity as it in itself defines the full fledged conditional coverage. As everything costs, cyclomatic complexity calculation costs good. So to assign initial number of clusters any other methodology may be considered that will provide coverage advantage as well as reliable guess for number of clusters.

So In the future, we will try to use any other technique for initial guess and get more optimized test case reduction. As a cluster may represent more than one condition/path, we can not say, test of a value for a cluster will give 100 % same result as a test of any other value in the same cluster. Also in the future, we will derive the effectiveness of reduced test suite empirically.

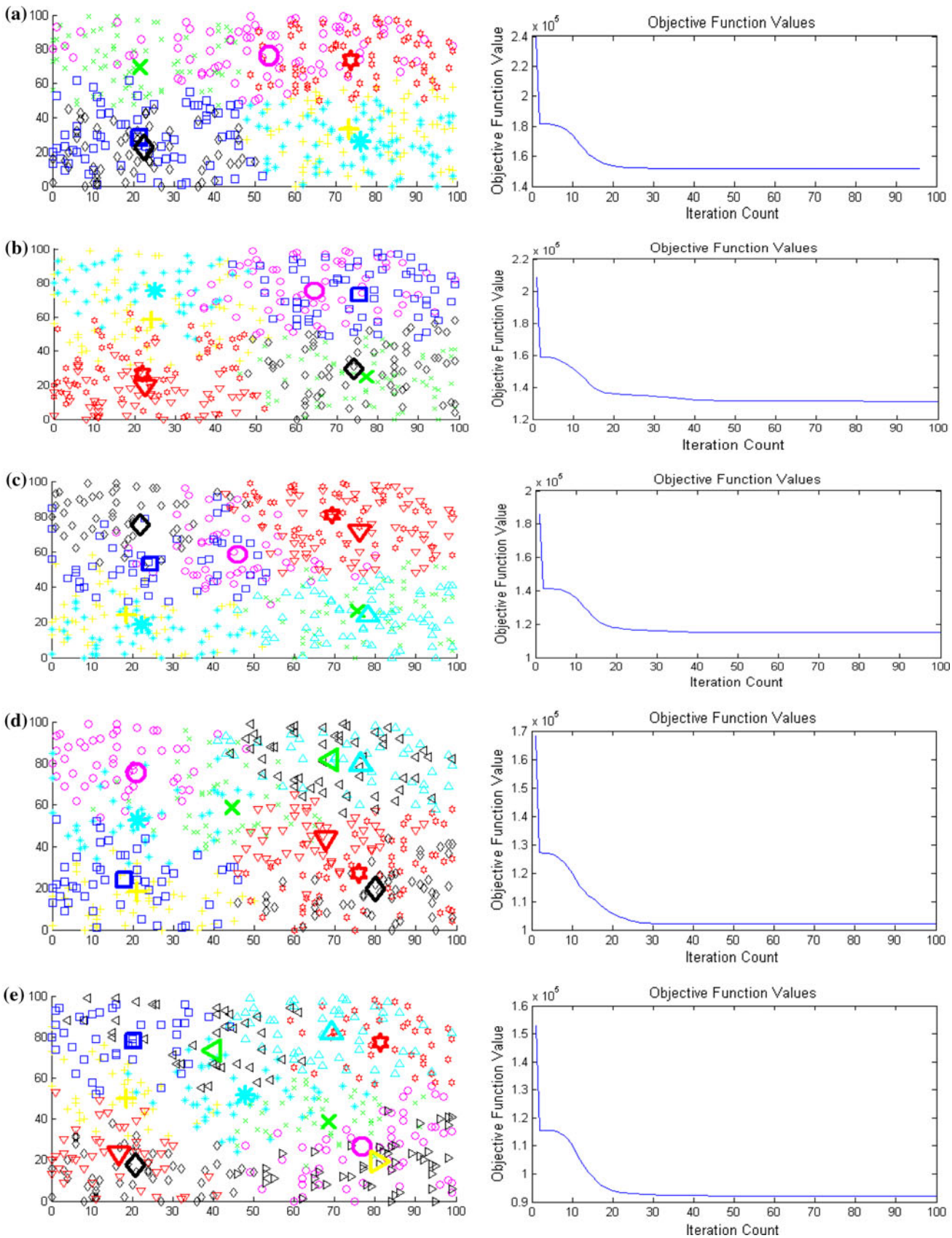

Fig. 1 Plot of clusters with their centers and objective function versus iteration count for 7, 8, 9, 10 and 11 numbers of clusters (**a**–**e**)

Table 3 Number of clusters and corresponding standard deviation

No. of clusters	Aggregate of cluster centers			Standard deviation
	Center1	Center2	Center3	
7	48.9506	47.1109	47.56913	0.957688
8	48.0441	47.8811	47.60591	0.221492
9	47.9966	48.0614	48.00642	0.03492
10	49.404	48.1169	46.69522	1.354922
11	49.1671	48.8998	46.26131	1.606047

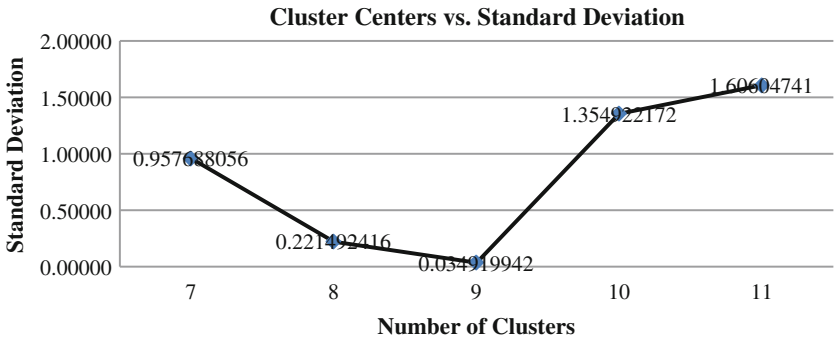

Fig. 2 No. of clusters w.r.t. standard deviation of cluster centroids

7 Conclusion

In this research paper, area of software testing and the size of test cases is studied and analyzed by considering a very simple and common example. Fuzzy Logic toolbox of MatLab is used for clustering the test cases in the form of data points. Initial number of clusters is assigned by calculating cyclomatic complexity for the program. Fuzzy is used for clustering to get more efficient and accurate results as compared to a generalized clustering algorithm. Hereby we proposed a new technique in software testing which can reduce the size of the test suite significantly. Also the reduced test suite may be tested for coverage of conditions/paths. We have also proposed future works for these outlines of research. So the use of the proposed methodology with fuzzy clustering in reducing test cases may significantly reduce time and effort spent in executing thousands of test cases.

References

1. Jones JA, Harrold MJ (2003) Test-suite reduction and prioritization for modified condition/decision coverage. IEEE Trans Softw Eng 29(3):195–209
2. Singh NP, Mishra R, Yadav RR (2011) Analytical review of test redundancy detection techniques. Int J Comput Appl (0975–8887) 27(1):30–33
3. Khan, Rehman, Malik (2009) The impact of test case reduction and prioritization on software testing effectiveness. Paper presented at the international conference on emerging technologies, Oct 2009, pp 416–421
4. Pringsulaka P, Daengdej J (2006) Coverall algorithm for test case reduction. Paper presented at the IEEE aerospace conference, p 8
5. Pan N, Zeng F, Huang Y-H (2010) Test case reduction based on program invariant and genetic algorithm. Paper presented at the 6th international conference on wireless communications networking and mobile computing (WiCOM), Sep 2010, pp 1–5
6. Askaruinisa, Abirami AM (2010) Test case reduction technique for semantic based web services. Int J Comput Sci Eng 02(03): 566–576
7. Muthyala K, Naidu R (2011) A novel approach to test suite reduction using data mining. Indian J Comput Sci Eng 2(3):500–505
8. Saraph P, Last M, Kandel A (2003) Test case generation and reduction by automated input-output analysis. IEEE Int Conf Syst Man Cybern 1:768–773
9. Dong W (2008) Test case reduction technique for BPEL-based testing. Paper presented at the international symposium on electronic commerce and security, pp 814–817
10. Raamesh L, Uma GV (2010) An efficient reduction method for test cases. Int J Eng Sci Technol 2(11):6611–6616
11. Copeland L (2004) A practitioner's guide to software test design. Artech House, Boston
12. Khan SR, Nadeem A, Awais A (2006) TestFilter: a statement-coverage based test case reduction technique, multitopic conference, Dec 2006, pp 275–280
13. Fraser G, Gargantini A (2009) Experiments on the test case length in specification based test case generation. Paper presented at the IEEE ICSE workshop on automation of software test, May 2009, pp 18–26
14. Myers GJ (2004) The art of software testing, 2nd edn. Wiley, Hoboken
15. Kaner C, Falk J, Nguyen HQ (1993) Testing computer software, 2nd edn. International Thomson Computer, Boston
16. McCabe T (2000) Cyclomatic complexity and the year 2000. IEEE Softw 13(3):115–117

17. Emergy KO, Mitchell BK (1989) Multi-level software testing based on cyclomatic complexity. In: Aerospace and electronics conference, Proceedings of the IEEE, May 1989, Vol. 2, pp 500–507
18. Wang L (2010) A program segmentation method for testing data generating based on path coverage, IEEE international conference on software engineering and service sciences, July 2010, pp 565–568
19. Akthar Shaheda, Rafi SkMd (2010) Improving the software architecture through fuzzy clustering technique. Indian J Comput Sci Eng 1(1):54–57
20. Khatibi BV, Jawawi DNA, Hashim SZM, Khatibi E (2011) A new fuzzy clustering based method to increase the accuracy of software development effort estimation. World Appl Sci J 14(9):1265–1275
21. Alavi R, Lotfi S (2011) The new approach for software testing using a genetic algorithm based on clustering initial test instances. Int Conf Comput Softw Model (IPCSIT) 14:225–231
22. Windham MP (1982) Cluster validity for the fuzzy c-means clustering algorithm. IEEE Trans Pattern Anal Mach Intell 4:357–363
23. Tilson LV, Excell PS, Green RJ (1988) A generalisation of the fuzzy c-means clustering algorithm. Paper presented at the international conference on geoscience and remote sensing, vol 3, pp 1783–1784
24. Davis JP, Warms TM, Winters WR (1991) A neural net implementation of the fuzzy c-means clustering algorithm. Int Jt Conf Neural Netw (IJCNN-91) 2(2):953
25. Groll L, Jakel J (2005) A new convergence proof of fuzzy c-means. IEEE Trans Fuzzy Syst 13(5):717–720
26. Yoo S, Harman M (2007) Regression testing minimisation, selection and prioritisation: a survey. Softw Test Verif Reliab, Wiley InterScience. doi:10.1002/000