Test Case Reduction Using Data Mining Classifier Techniques

Ahmad A. Saifan*

Department of Computer Information Systems, Faculty of Information Technology and Computer Sciences, Yarmouk University, Irbid, Jordan.

* Corresponding author. Tel.: +962775646005; email: ahmads@yu.edu.jo
Manuscript submitted February 4, 2016; accepted March 4, 2016.
doi: 10.17706/jsw.11.7.656-663

Abstract: The main purpose of test case reduction is to decrease the number of test cases in order to minimize the time and cost of executing them. We used the data mining approach, mainly because of its ability to extract patterns of test cases that are invisible. In this paper, we use two data mining classifiers, Naïve based and J48, in order to classify a set of test cases and identify which test case is redundant or irredundant. The results show the applicability of data mining classification in removing the number of redundant test cases.

Key words: Software testing, test case reduction, redundant test cases, naïve based classifier, J48 classifier.

1. Introduction

The purpose of software testing is to provide an objective view of software to all users and stakeholders to predict the future quality of the system. There are two methods of software testing, white box testing and black box testing [1]. These two methods use a number of techniques to perform software testing. White box testing uses three methods that focus on finding defects in software in order to test specific software: path testing, loop testing, and control structure testing. Black box testing, on the other hand, uses other more techniques for testing such as fuzz testing, model based testing and basic path. All these methods are described in [1].

The size and complexity of software systems is growing dramatically. In addition to this, the existence of automated tools has led to the generation of a huge number of test cases, the execution of which causes huge losses in cost and time [2]. According to Rothermel et al. [3], a product of about 20,000 lines of code requires seven weeks to run all its test cases. Ultimately, the challenge is to find a way to reduce the number of test cases or to order the test cases to validate the system being tested. The main goal of software testing is ensure that the software is as free as possible from errors. The test process can be said to be effective when the test cases are able to locate any errors. Several tools have been seen in the literature which automatically generate thousands of test cases for a simple program in a few seconds, but executing these test cases is extremely time-consuming. Moreover, the tools may also generate redundant test cases [4].

The problem is compounded when we have complex systems, where the execution of the test cases may take several days to complete. Moreover, it should be noted that most of the time is spent in executing redundant or unnecessary test cases. For all of the aforementioned reasons, software testing uses certain techniques to reduce the number of test cases that are both time- and cost-effective which directly affect software performance, for example; retirement optimization, DDR (dynamic domain reduction), Ping Pong, CBR (case based reasoning), and coverall algorithm.

This paper aims to deal with this issue of reducing the number of test cases in order to minimize the time and cost of executing them. Several techniques can be used to reduce test cases including information retrieval, pairwise testing and data mining clustering techniques such as k-mean. In this paper, we used J48 and Naïve based as classification methods for the software dataset. The aim of these classifiers is to reduce any ambiguity in the dataset by dividing it into specific classes. We use these classifications to identify the redundant test cases in the dataset. Based on our knowledge little work has been done in this area. The rest of the paper is organized as follows: Section II formally describes our test case reduction technique. Section III discusses background information and related work. The methodology, experiment and the analysis results are described in Section IV. Finally, Section V presents the conclusions.

2. Problem Description

Given a program $Prog = \{P_1, P2, \dots, P_n\}$ consists of a set of packages P, a package that consists of a set of classes $P = \{C_1, C_2, \dots, C_m\}$, a class consists of set of methods $C = \{M_1, M_2 \dots, M_k\}$.

A set of test cases is defined to test the program for bugs including class, methods, number of inputs and their types, excepted outputs, and status.. The set of test cases is defined as $T = \{t_1, t_2, \dots, t_\propto\}$, where $t_i = \{a_1, a_2, \dots, a_t\}$ and $Type$(a) $\subseteq$ {Integer, string, double, boolean, char, byte} is a test case defined to cover one of the following:

Class Coverage: $Ccov(t_i)$=$\{C_1, C_2, \dots, C_{m1}\} \subseteq C$ covered by t_i

Method Coverage: $Mcov(t_i) = \{M_1, M_2, \dots, M_{k1}\} \subseteq$ M $called\ by\ t_i$

Input coverage: $Icov(t_i)$ =$|a|$ covered by t_i

In order to reduce the number of generated test cases, two classifiers are used to determine whether a test case is redundant or irrredundant. The two approaches are Naïve based and J48 decision tree. To predict the status of the test cases we followed the following three rules. Specifically, a test cases t_i is considered redundant with t_j if it is satisfied with all of the three rules, otherwise; the test case t_i is considered to be irredundant. The three rules are:

$\forall\ (i,j)\ where\ i \neq j\ , \Big(Ccov(t_i) = Ccov\big(t_j\big)\Big) \subseteq C$

$\forall\ (i,j)\ where\ i \neq j\ , \Big(Mcov(t_i) = Mcov\big(t_j\big)\Big) \subseteq M$

$\forall\ (i,j)\ where\ i \neq j\ ,\ Icov(t_i) = Icov\big(t_j\big)$

3. Background and Related Work

Many studies show that there are two types of test case reduction technique: (a) pre-processing and (b) post-processing. Pre-processing reduces the number of test cases immediately after creating the test set. Post-processing maintains and removes unnecessary test cases, after running the first regression testing activities. As we have described above, several techniques are proposed in the literature for test case reduction (also known as redundant test cases), based on several algorithms such as requirements optimization, DDR (Dynamic Domain Reduction), Ping Pong, CBR (Case Based Reasoning) and coverall algorithm. The aim of these techniques is to minimize and remove redundant test cases. Below, we describe the algorithms in detail.

3.1. Get Split Algorithm [5]

This is an algorithm within Dynamic Domain Reduction (DDR) used to split a domain by determining the split point to obtain new domains that have two variables with constraints. The two new domains must satisfy the constraints, and the size of the two new domains should be equal. There are two cases for two variables that have been modified for new domains. These cases are defined depending on the relationships between the two variables' domains.

1) Non-intersecting sets of values defined by two domains, in which case the constraint is either satisfied, or is infeasible.
2) Intersecting sets of values defined by two domains, in which case the constraint may or may not be satisfied.

The purpose of Get Split is to satisfy a constraint for all pairs of values of the two domains, and be true for all values.

3.2. Ping Pong [6]

Ping Pong is a technique proposed to select fewer test cases by reordering test cases, based on heuristics technique. Ping Pong can follow two procedures. Assuming we have a test set *T* of size *N*, ordered in sequence select = {*t*1, *t*2, *t*3 ... *tN*}. The procedures are:

1) Forward Procedure: execute test cases starting from t1 to *tN* then deduct those which are ineffective to obtain a smaller list of test cases .
2) Reverse Procedure: execute test cases in reverse order from *tN* to *t*1, then deduct those which are ineffective to obtain a smaller list of test cases .
3) Inside Out Procedure: execute test cases from the middle to both ends, if *N* is odd we run *t*(*N*+1)/2 ,*t*(*N*+1)/2-1, *t*(*N*+1)/2 +1 ,....,*t*1, *tN*, on the other hand if *N* is even we run *tN*/2 +1, *tN*, *tN*-1,....,*t*1, *tN*, then deduct those which are ineffective to obtain a smaller list of test cases.

3.3. CBR [7]

Case based reasoning (CBR) technique is one of the most famous techniques and is designed with artificial intelligence. CBR mainly consists of four phases: retrieve, ruse, revise, and retain [8]. CBR depends on problems that are similar to the new problem on the generation process of test cases.

3.4. Coverall Algorithm [8]

Coverall Algorithm follows certain rules and has its own domain like other algorithms. Coverall algorithm generates cases in four steps:

Table 1. A Comparison between Test Case Reduction Techniques

Technique	Process	Coverage	Cost and performance
CBR	-The CBR still has a problem in time reducing, and there are many techniques used in the CBR that help to control the test case number - The CBR deletion algorithm takes a long time to delete all the redundancy test case [9]	Path coverage method or criteria to minimize the test case redundancy	-Very important to improve the performance by using the case-base maintenance -In spite of many reduction techniques, CBR has a high cost
Ping- Pong	-Doesn't give the best solution but give the best solution in appropriate Time [6] -Require execution of the program on each test case at least once, and once for each testing requirement, thus lead to time consuming in some cases	- Reduce statement coverage based test sets by 50%[10] -Assure domain Coverage [6]	-Mutation based test sets reduced by over 30 this result as explained through experiment in paper [2] -This technique can be expected to be cost-effective
Coverall algorithm	Lower time in test run and compilation. - Comparing to all other techniques it is the best one [6]	Reduce number of test cases by more than 99% [8]	The technique can also be used in program loops and arrays [8]
Get Split	The least time-consuming among the existing technique[10]	Achieved a greater reduction percentage of the test cases	-The technique can also handle loops and array

		but lower than coverall	dynamically [10] - Still have problems in handling pointers , less effective than others

Finding all possible constraints (where constraints are algebraic expressions that restrict the space of program variable to a certain domain) that dictate the conditions of variables between start and end nodes (<,>,=,≥,≠), which mean from the initial point to the final point .

Identifying and finding fixed values for variable as maximum, minimum and constant values in the path, this requirement may be consider a disadvantage of the techniques. Using the condition dictated by constraint, two variables, one with maximum and the other with minimum value, can be identified. Then the maximum variable is set at the highest value within its range and minimum variable at the lowest value of its range. If any constant value for any variable is found, the value is assigned to a constant value detected in the path .

Finding constant values in the path for any variable, the values would then be assigned to the given variables at each node.

Using all of the abovementioned values to create a table to present all possible test cases.

A comparison between test case reduction techniques

Table 1 shows a comparison between the different test case reduction techniques described above. The comparison is based on three different criteria: the process of the technique, the supported coverage criteria, and the cost and the performance of applying the technique. Some other test case reduction techniques have been proposed in the literature, including heuristic algorithms [11], genetic algorithm based approach [12], integer linear programming approach [13], and hybrid approach [14]. A survey proposed in [14] summarizes these techniques.

3.5. Data Mining

There are several works in the literature on reducing the number of test cases using data mining techniques. Some of these use clustering technique such as that presented in [4], [15]. They propose a novel model for test suite reduction using clustering data mining technique based on software coverage; their model generates test cases automatically and saves them in a WEKA file format (arff), then they apply k-mean clustering on that file and save the cluster results in a text file. Then, the file is loaded in pickup clustering function then the algorithm is executed and the output saved to a text file. Finally, the text files are used to test for coverage of the software and these steps are repeated until acceptable coverage is achieved. The result shows that this methodology significantly reduces time and effort spent in executing thousands of test cases. On the other hand, some other studies use different classifiers [5], [16], [17], [18] in different domains using tools such as WEKA, Rapid Miner and ORANGE. Lee and Chan [19] proposed an approach that can be used to enhance the process of automated test tool model using machine learning technique from association rule mining.

In this paper we propose a classification technique in order to reduce the number of test cases. We used WEKA to define our training data set. Usually, the given data set is divided into a training set and a testing set; the training set is used to build the model and the test set (test case) is used to validate it. Test cases normally consist of a unique identifier, requiring references from a design specification, preconditions, events, a series of steps (also known as actions) which should be followed, input, output, expected result, and actual result. Reducing the number of test cases can improve test runtime and simplify debugging when an error is found [2] .

4. Methodology

Our test case reduction approach consists of three steps. We start by obtaining and generating the test cases for the system being tested. Then, we build our dataset for the test cases based on the most important attributes of the test cases. Finally, we select the classifier and rules that we are going to apply and extract the results. Below we describe each of these steps in detail.

4.1. Test Case Generation for the System under Test

Our approach starts by selecting the source code. In this paper, we select a Java source called "Cinema". Table 2 shows some of the properties of the source code such as the system classes, lines of code, and the number of methods in each class. In order to run the system we use eclipse SDK 3.7.2 software which is an integrated development environment (IDE) which contains a base workspace and an extensible plug-in system for customizing the environment written mostly in Java programming language.

Table 2. Cinema System Classes, Lines of code and Number of Methods

Class name	Lines of code	Number of methods
Cine	338	16
Sala-infantil	205	8
Sala-vip	148	11
Session	46	9
Asiento	71	15
Sala	77	25
Total	885	84

In this paper we use one of the key features of this tool which is test case generation using JUnit. The tool is used to automatically generate a set of test cases for any given input class. We are able to generate 493 test cases for all the above classes.

4.2. Build Data Set

Our data set consists of several attributes. Table 3 describes all these attributes. To obtain the attribute values for all the test cases we use several tools and plugins. For example, we use Eclemma [20] a plug-in in Eclipse to calculate the values of coverage for class, path, methods, and branches attributes for each test case. For TC_RunTimeDuration attribute, we use DJunit [21] a plugin in Eclipse to calculate runtime needed to execute each test case. The last tool, we use is JaCoCo Metrics [20] a plugin in Eclemma to calculate the cyclomatic complexity for each test case. Finally, the inputs, outputs, and status are given for each test case. The Data set contains 493 test cases with 10 attributes and Class label. The test cases consist of 268 irredundant and 225 redundant cases, which indicates that we have a balanced dataset. For any dataset sometimes it is necessary to undertake some preprocessing steps, and in our project we use ReplaceMissingValue Filter to replace the missing value by mean value of other instances in the same attribute. Many tools have been proposed for the data mining field such as ORANGE, RAPID MINER and WEKA. We chose WEKA because it has more than one type of classifier and it has a maximum amount of accuracy especially when we select the best classifier. Naïve Bayes classifier has almost maximum accuracy when it is applied to different data sets [22].

In this approach we have applied the machine learning techniques on WEKA to our dataset in order to generate several patterns and rules. These data mining techniques involve Naïve based and J48 decision tree to decide whether each test case is redundant or irredundant.

As described in Section 2 the following are the rules used to predict whether or not a test case is redundant:

Table 3. Data Set Attributes

Attribute Name	Attribute Description	Data type
TC_ID	A sequential idintifier for a test case	Numeric
TC_ClassCoverage	The class number covered by the test case. We have 6 classes, TC_ClassCoverage range from 1 to 6	Numeric
TC_PathCoverage	The number of paths covered by the test case	Numeric
TC_MethodCoverage	The number of methods covered by the test case	Numeric
TC_BrancheCoverage	The number of branches covered by the test cases.	Numeric
TC_RunTimeDurartion	The time needed to executate the test case.	Numeric
TC_Input	The inputs of the test case	String
TC_Output	The expected outputs of the test case	String
TC_Status	An Indicator to represent that test case can reveal faults or not... If the status is fail, it mean that fault is detected.	Boolean [0 , Fail][1, Pass]
TC_CycomaticComplexity	The cyclomatic complexity of the test case.	Numeric
Class label	An indicator of Class label	Redundant=0 Irredundant=1

What is the component tested by the test case?

What is the function tested by the test case?

How many inputs are there, and is the type of each input?

The following Table 4 shows a sample of the dataset that is used in our approach.

Table 4. Sample of Data Set

TC_ID	TC_ClassCoverage	TC_MethodCoverage	TC_Input	Class Label
1	5	8	2,4,"234"	Irredundant
2	5	8	1,4,"a23"	Redundant
3	5	8	2,4,"a2c"	Redundant

As we can see from this table we have 3 test cases. Test case with TC_ID=1 has the following values:

1) Cover class number = 5
2) Cover method number =8
3) Three inputs = 2,4,"234"
4) An irredundant class

Because the above three test cases all cover the same class, the same method and have the same number of inputs, then the first is irredundant and the other two are considered to be redundant.

4.3. Extract Results

In this phase we applied Naïve based and J48 classifier to the dataset that we built in order to compare the results from the two classifiers. After that, we used WEKA to extract the accuracy of the classification. Table 5 shows the accuracy for the two classifiers before and after the preprocessing step with different validation folds, when the values of the class label are "redundant and irredundant".

As we can see from TABLE V, the Naïve base gives correctly classified instances before preprocessing with 61.74% for 11 folds. After removing empty instances and the attributes that have less information, the accuracy increases to 70.31% for 11 folds.

On the other hand J48 give correctly classified instances before preprocessing with 58.62% for 10 folds. But after removing empty instances and the attributes that have less information, the accuracy decreases to 57.14%.

The results show the applicability of data mining classification techniques to classify whether the test case is redundant or irredundant based on a set of rules built on a set of characteristics of the test cases.

Table 5. Naïve Base and J48 Comparison

# of folds	Naïve base		J48	
	Accuracy (without Preprocessing)	Accuracy (with Preprocessing)	Accuracy (without Preprocessing)	Accuracy (with Preprocessing)
5	56.18	61.34	55.83	54.73
7	58.60	68.12	53.16	51.16
10	57.32	62.25	58.62	55.62
11	61.74	70.31	56.34	56.42
13	57.32	65.46	57.14	57.14
15	59.86	68.12	53.16	52.86

5. Conclusion

The paper focused on the applicability of data mining classifier techniques in reducing the number of test cases by removing those which are redundant. The J48 and Naïve base were applied to the dataset in order to build the classifiers. For future work we try to show the ability of the proposed technique in revealing faults in the software under test.

References

[1] Jovanović, I. (2006). Software testing methods and techniques. *The IPSI BgD Transactions on Internet Research, 30*.

[2] Raamesh, L., & Uma, G. V. (2010). Reliable mining of automatically generated test cases from software requirements specification (SRS). *IJCSI International Journal of Computer Science Issues*, *7(3)*.

[3] Rothermel, G., Untch, R. H., Chu, C., & Harrold, M. J. (2001). Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering*, *27(10)*, 929-948.

[4] Muthyala, K., & Naidu, R. (2011). A novel approach to test suite reduction using data mining. *Indian Journal of Computer Science and Engineering*, *2(3)*, 500-505.

[5] Offutt, A. J., Jin, Z., & Pan, J. (1999). The dynamic domain reduction procedure for test data generation. *Software-Practice and Experience*, *29(2)*, 167-193.

[6] Salwan, R., & Sehgal, R. (2010). Test cases reduction technique considering the time and cost as evaluation slandered. *International Journal of Computer Science and Its Application*.

[7] Roongruangsuwan, S., & Daengdej, J. (2010). Test case reduction methods by using CBR. *International Workshop on Design, Evaluation and Refinement of Intelligent Systems (DERIS2010)* (p. 75).

[8] Pringsulaka, P., & Daengdej, J. (2005). Coverall algorithm for test case reduction.

[9] Gupta, A., Mishra, N., & Kushwaha, D. S. (2014, February). Rule based test case reduction technique using decision table. *Proceedings of the Conference on 2014 IEEE International Advance Computing* (pp. 1398-1405).

[10] Mahapatra, R. P., & Singh, J. (2008). Improving the effectiveness of software testing through test case reduction. *World Academy of Science, Engineering and Technology*, *37*, 345-350.

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

[12] Mansour, N., & El-Fakih, K. (1999). Simulated annealing and genetic algorithms for optimal regression testing. *Journal of Software Maintenance*, *11(1)*, 19-34.

[13] Black, J., Melachrinoudis, E., & Kaeli, D. (2004, May). Bi-criteria models for all-uses test suite reduction. *Proceedings of the 26th International Conference on Software Engineering* (pp. 106-115).

[14] Yoo, S., & Harman, M. (2010). Using hybrid algorithm for pareto efficient multi-objective test suite

minimisation. *Journal of Systems and Software*, *83(4)*, 689-701.

[15] Kameswari, U. J., Saikiran, A., Reddy, K. V. K., & Varun, N (2011). Novel techniques for test suite reduction. *International Journal of Science and Advanced Technology*, *1(8)*.

[16] Rahman, R. M., & Afroz, F. (2013). Comparison of various classification techniques using different data mining tools for diabetes diagnosis. *Journal of Software Engineering and Applications*, *6(03)*.

[17] Fassnacht, F. E., Neumann, C., Forster, M., Buddenbaum, H., Ghosh, A., Clasen, A., & Koch, B. (2014). Comparison of feature reduction algorithms for classifying tree species with hyperspectral data on three central European test sites. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, *7(6)*, 2547-2561.

[18] Zeng, F., Li, L., Li, J., & Wang, X. (2009, June). Research on test suite reduction using attribute relevance analysis. *Proceedings of the Eighth IEEE/ACIS International Conference on Computer and Information Science* (pp. 961-966).

[19] Lee, E. M., & Chan, K. C. (2005). A data mining approach to discover temporal relationship among performance metrics from automated testing. *Proceedings of the Ninth IASTED International Conference on Software Engineering and Applications.*

[20] Eclemma, Java code coverage for eclipse (2016). Retrieved April 9, 2016, from: http://www.eclemma.org/index.html

[21] DjUnit plugin for Eclipse. (2016). Retrieved April 9, 2016, from: http://works.dgic.co.jp/djunit/

[22] Garcıa-Saiz, D., & Zorrilla, M. E. (2011). Comparing classification methods for predicting distance students' performance. *Proceedings of the International Workshop on Applications of Pattern Analysis*.

Ahmad Saifan is an assistant professor in the Department of Computer Information Systems at Yarmouk University (YU) in Jordan. He obtained his Ph.D degree in software engineering from Queen's University (Canada). His master degree in computer science from YU. He had his B.Sc degree in computer science from YU. His research interest include are model-based testing, regression testing, software testing.