ISSN : 0974 - 7435 Volume 10 Issue 24

2014

BioTechnology

An Indian Journal

FULL PAPER

BTAIJ, 10(24), 2014 [16500-16506]

Test suite reduction for mutation testing based on formal concept analysis

Liping Li[1*], Xingsen Li[2]

[1]Computer and Information Institute, Shanghai Second Polytechnic University, Shanghai, (CHINA)

[2]Management School, Ningbo Institute of Technology, Zhejiang University, Ningbo, (CHINA)

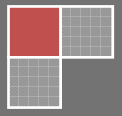
E-mail: liliping@sspu.edu.cn, lixs@nit.zju.edu.cn

ABSTRACT

Formal concept analysis (FCA) is a method used for deriving implicit relationships between objects by attributes. Aim at the expensive cost problem in mutation testing caused by the large number of mutants and large number of test cases generated to kill these mutants. This paper proposed a test suite reduce method for mutation testing based on FCA. In order to reduce the number of test cases, a test generation algorithm and three reduction rules were presented to reduce the set of test suite based on concept lattice. Results showed this approach can help to reduce the redundancy of test cases for mutation testing in some degree.

KEYWORDS

Government financial information; Disclosure quality; Factors; Consequences.



INTRODUCTION

Mutation testing is a fault-based software testing technique[1]. It has been empirically found to be the most effective in detecting faults amongst the various testing strategies[2]. Mutation testing has been successfully applied to many programming languages as a white box unit testing technique[3]. It also applied to specification testing and test data generation etc.

But, one of the major problems of the mutation testing is the expensive cost incurred by so many mutants and test cases generated to kill these mutants. Mutation testing is generally regarded as too expensive to use in reality. There are lots of mutants even for small programs. For that, various cost reduction techniques have been proposed to reduce the number of mutants, such as Mutant Clustering[3, 4], Mutant Sampling, Selective Mutation[5] and Higher order Mutation[6] etc.

In this paper, we research test suite reduction for mutation testing. We first cluster the similar mutants to form mutant clustering based on similar mutants. Then, refer to the work of[7], we introduce some reduction rules to minimize the number of test cases which used to kill the mutants.

The remainder of this paper is organized as follows: Section 2 introduces the concept of mutation testing and the formal concept analysis which used to reduce the test suite. Section 3 presents an algorithm of test generation for mutation testing. Section 4 discusses the approach for test suite reduction based on concept lattice. Section 5 presents the experiments study. Section 6 summarizes some related work and at the end concludes the paper.

THE PRELIMINARIES

Mutation testing

Mutation testing first making small changes to the original program, the changes are called *mutants*, then running all mutants against all current test cases in order to killing the mutants. Most mutation systems introduce one change at a time. After execution, the mutation score is calculated to measure the effectiveness of the test suite for its ability to detect faults.

Usually, mutation testing involves three steps:

1) **Mutant generation**. Mutation operators are applied to the original program to get a set of mutants.
2) **Mutant execution**. Test cases are executed against the original program and all the mutants.
3) **Result analysis**. Results of the executions are analyzed and the mutation score is calculated.

A mutation operator is a rule that is applied to a program or system under test (SUT) to create mutants. Typical mutation operators, for instance, replace each operand by every other syntactically legal operand, or modify expressions by replacing operators and inserting new operators, or delete entire statements etc. For example, expression (x/y) being mutated to (x%y) is belong to Arithmetic Operator Replacement (AOR). Design effective and efficient mutation operators are one of the key problems of mutation testing. Researchers in this field have proposed many kinds of mutation operators. TABLE 1 lists five typical operators and present some details about them. These five operators are generally considered as the most effective operators in mutation testing[1]. It is worth noticing that even an operator will possibly generate many mutants, because there are not only different feasible replaceable actions but also different locations in a program for change[2].

TABLE 1 : Five typical mutation operators

Mutation operator	Description	Feasible action
AOR	Arithmetic operator replacement	+, -, *, /, %
ABS	Absolute value insertion	|x|
LCR	Logical connector replacement	&&, ||
ROR	Relational operator replacement	<, <=, >, >=, = =, !=
UOI	Unary operator insertion	++, --, !, +, -

There exist some tools to generate mutants automatically, such as, Mothra[4] for FORTRAN programs, Proteum[5] for C programs, and MuJava[12] for Java programs. This paper uses MuJava to generate mutants for Java programs, and clustering the similar mutants to form mutant clustering based on mutant distance.

In step 3 of mutation testing, if the running result of the original program and all the mutants is different, then the fault is detected and the mutant is said to have been killed. If mutant p' is different with original program p in grammar, but consistency with p in semantic, then called p' is the equivalent mutant of p. Equivalent mutants are mutants that produce the same output as the original program, so cannot be killed.

The mutation score can be considered as a testing criterion which can be used to measure the effectiveness of a test suite in terms of its ability to detect faults. The formula of mutation score[3] is shown as follows:

$$MS(S, T) = K / (M-E)$$

Where, *S*: System under test, *T*: Test cases, *M*: Number of mutants, *K*: Number of killed mutants, *E*: Number of equivalent mutants.

Ideally, the goal of tester is to raise the mutation score to 1.00, indicating the test suite T is sufficient to detect all the faults denoted by the mutants. A test case must fulfill three conditions to kill a mutant[1]:

1) **Reachability**. The mutated statement of the system must be executed by the test case.
2) **Infection**. The execution of the mutated statement must put the mutant into an erroneous state.
3) **Propagation**. The erroneous state must propagate until it reaches the external environment.

Formal concept analysis

Formal concept analysis (FCA)[7-9] is a method used for deriving implicit relationships between objects based on attributes. FCA is a hierarchical clustering technique[3] for objects with discrete attributes. Through FCA, we can systematically identify similarities and differences by constructing a hierarchy of object groups. FCA can be used to cluster and order mutants in mutation-based test case generation[9]. FCA can also be used for test suite minimization[7]. The main notion of FCA is formal context.

Definition 1 (Formal Context). A formal context is a 3-tuplet (O, M, R) if O is an object set and M is an attributes set, and $R \subseteq O \times M$ is a binary relation between O and M.

For $X \subseteq O$, X' is the set of all attributes common to the objects of X, if $X' = \{m \in M | \ o \in X: (o,m) \in R\}$

For $Y \subseteq M$, Y' is the set of all objects that have all attributes in Y, if $Y' = \{o \in O | \ m \in Y: (o,m) \in R\}$.

In this paper, a set O of objects represent test cases, a set M of attributes denote the mutants. Paper[9] also uses FCA for test generation, in their work, objects represent mutants, and attributes denote the mutation operator.

Definition 2 (Formal Concept). A pair (X, Y) is a formal concept of (O, M, R) if and only if $X \subseteq O$, $Y \subseteq M$, $X'=Y$ and $Y' = X$.

That is mean, (X,Y) is a formal concept if the set of all attributes shared by the objects of X is identical with Y and on the other hand X is also the set of all objects that have all attributes in Y. X is then called the extent and Y the intent of the concept. The formal concepts of a given context are ordered by the partial order $\leq$ defined by:

$$(X, Y) \leq (X', Y') \equiv X \subseteq X' \ (\equiv Y' \subseteq Y)$$

Which gives the set of all concepts for a formal context (O, M, R) the structure of a complete lattice.

Figure 1 is a simple example referred to[9] of formal context and the resulting concept lattice. The top concept denotes the set of all objects and the set of attributes common to all objects, and the bottom concept represents the set of all attributes and the set of objects that includes all attributes.

In this paper, objects represent test cases and attributes denote mutants which can be killed by the test cases. We should find out the implicit relationship between mutants and its attributes. The relationship R between test cases and mutants is defined by: R = {(TS, Mu) $\in O \times M$ | ts *kills* mu}. Mutants that are killed by the same test case are clustered together to form the concept lattices by the context (O, M, R)

Object	Attributes
A	a,b,c
B	b,d,e
C	f,g

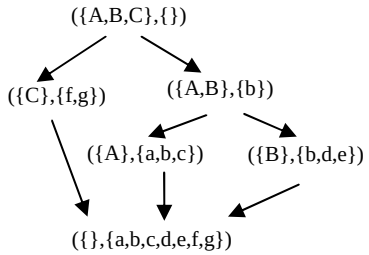


Figure 1. A Demonstration of Concept Lattice

TEST GENERATION FOR MUTANT TESTING

In mutation testing, some representative errors are deliberately seeded into the SUT (System under Testing) to create a set of mutants, and run all current test cases on all mutants. There are so many mutants even for little system, for example, the triangle program with 50 lines of code, can have 584 mutants[2]. Equivalent mutants are mutants that cannot be killed. Determining equivalent mutants automatically are also an important way to reduce the cost and promote the acceptance of

mutation testing[6]. In this paper, we clustering mutants based on similar mutants who distance are close enough and proposed a test generation algorithm for mutation testing.

At first, we apply MuJava[12] to generate mutants M for system S. Second, we find all variables in S. The domains of the variables are recorded according to the characteristic of every mutant. We take the domain as an attribute of every mutant. Third, we calculate the distance between every two mutants. The type of distance between every two mutants includes Hamming distance and Euclidean distance. Refer to paper[14], mutants distance in this paper is an index describing the semantic difference between the original and the mutated program. A threshold value that is half of max distance value is specified. Randomly selects 2 mutants as the initial clusters. Cluster the mutants which have the distance less than the threshold value. Repeat the step 3, until the distance of all mutants large than the threshold value.

The test generation algorithm is shown as Figure 2. We input a system S and a set of mutants M, output a test suite T for the system S and mutants exclude equivalent mutants as a result. In line 1, T is the test suite; EM is the set of equivalent mutant. In line 2, K is the number of killed mutants, E is the number of equivalent mutants. We clustering mutants Mu based on similar mutants, show as line 3. We use the clustering algorithm of paper[3]. While Mu is not empty, select a mutant randomly from the Mu. Generate test case t to kill m, if m is killed, that is mean m is not an equivalent mutant, then t is merged into test suite T and m is removed from Mu, the number of killed mutant K plus one. If m is not killed, the number of equivalent mutants E adds one. Calculate mutation score MS(S, T) = K / (M-E), the threshold value for mutation testing we choose 0.9. If MS(S, T)>0.9, then we stop generate test cases, and the remained mutants in Mu are equivalent mutants which can not to be killed at present. Remove these equivalent mutants from M, we can get a smaller mutants set. Output test suite T and the new mutants set M which removed the equivalent mutants.

TEST SUITE REDUCTION BASED ON FCA

Test suite minimization problem is NP complete. Lots of researchers use the greedy heuristic algorithm to solve this problem in some degree. They first pick the test case which cover the most requirements, throw out all the requirements covered by the test case. Repeat the process until all requirements are covered. Formal concept analysis (FCA) can be used for deriving implicit relationships between objects based on attributes. Paper[7] used FCA for test suite minimization. Refer to their work, we also define some reduction rules to reduce the set of test suites based on FCA, call it TSM-FCA.

Test generation algorithm for mutation testing

Input: A system S and a set M of mutants
Output: A test-suite T for the system S, and a new set M of mutants except for the equivalent mutants.

```
1 T   ∅;
2 K  0; E  0;
3 Cluster mutants Mu⊆ M;
4 While Mu ≠∅ do
5 Pick mutant m   Mu;
6 Generate test case t to kill m;
7 if m is not an equivalent mutant
8 T   T   {t};
9 Mu    Mu \ {m};
10 K++;
11 else
12 E++;
13 endif
14 Calculate mutation score MS(S, T);
15 if MS(S, T)>0.9, then
16 break;
17 endif
18 endwihile
19 M    M \ Mu;
20 Output M;
```

Figure 2 : Test generation algorithm for mutation testing

TABLE 1 is an example showing test cases kill the mutants, e.g., test case t1 can kill mutants m1, m2, m3. If we use the classical greedy heuristic algorithm to minimize test suite, we will first select the test case t1 because it killed three mutants. And throw out the mutants m1, m2, m3. Then we can pick t2, t3, t4, t5 because each of them kills one yet un-killed mutant. If t2 is selected, then m4 is thrown out. Next, we select t3, and m5 is thrown out. Finally, t4 is selected to kill all mutants. Thus, the minimized test suite is {t1, t2, t3, t4}. Actually, the minimized test suite is {t2, t3, t4}. Test case t1 which was selected first was redundant for this example. And it also shows the drawback of the general greedy heuristic algorithm.

TABLE 1 : An example for mutation testing

	m1	m2	m3	m4	m5	m6
t1	X	X	X			
t2	X			X		
t3		X			X	
t4			X			X
t5					X	

Concept Analysis identifies maximal groupings of objects and attributes called concepts. In TABLE 1, the set {t1, t2} is the maximal set of test cases that kills the mutant m1. {t1} is the maximal set of test cases that kills all of the mutants m1, m2, m3. Refer to definition 2, if (X, Y) and (X', Y') are two concepts which have the partial order relationship (X, Y) ≤ (X', Y'), then X⊆X' and Y'⊆Y. In TABLE 1, concepts satisfy ({t1}, {m1, m2, m3}) ≤ ({t1, t2}, {m1}), since {t1} ⊆ {t1, t2} and {m1} ⊆ {m1, m2, m3}. The concept lattice is shown as Figure 3.

If the top concept of the resulting concept lattice happens to have a non-empty intent, any input taken from the intent is an optimal test case that kills all mutants in O. If the intent of the top concept is empty, no single test case suffices, but we can still extract test cases killing many mutants simultaneously from the maximal concepts with non-empty intent. But, the concept lattices generated by the context (O, M, R) are too idea to be computed in practice.

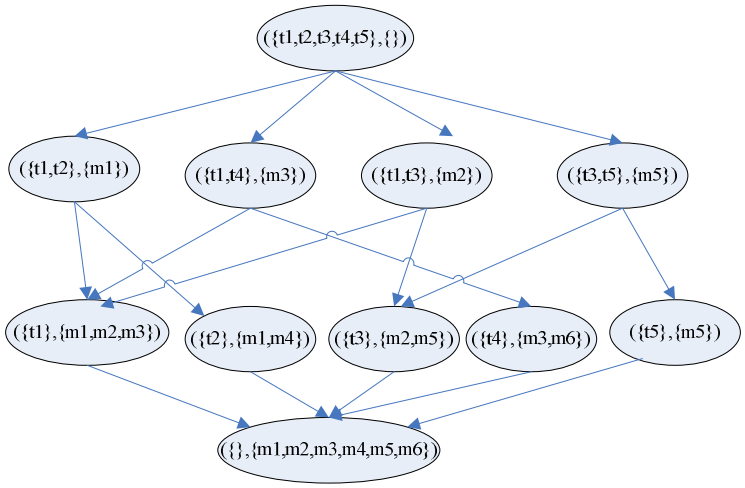


Figure 3 : The concept lattice for the example

In order to reduce the redundancy test cases, refer to paper[7], we defined two implications and three reduction rules based on FCA.

Object Implication: Given two objects o_1, o_2 O, $o_1 \Rightarrow o_2$ if and only if m M, (o_2 R m) => (o_1 R m).

Object reduction rule: For two objects o_1, o_2, if $o_1 \Rightarrow o_2$ then all the mutants killed by o_2 are also killed by o_1. So, the row corresponding to the object o_2 can be removed from the context table without affecting the effect of testing.

Attribute Implication: Given two attributes m_1, m_2 M, $m_1 \Rightarrow m_2$ if and only if o O, (o R m_1) => (o R m_2).

Attribute reduction rule: For two attributes m_1, m_2, if $m_1 \Rightarrow m_2$ then test case kill m_1 are also kill m_2. So, the column corresponding to the attribute m_2 can be removed from the context table without affecting the effect of testing.

In TABLE 1, there has an object implication t3=>t5, so we can remove row t5 because all the mutants killed by t5 are also killed by t3. We also can remove columns m1 and m3 by adopting the attributes reduction m4=>m1 and m6=>m3. The reduced context table is shown as TABLE 2 after applying objects reduction t3=>t5 and attributes reduction m6=>m3, m4=>m1.

TABLE 2 : Reduced context table of TABLE 1

	m2	m4	m5	m6
t1	X			
t2		X		
t3	X		X	
t4				X

In TABLE 2, there only has an object implication t3=>t1, so we can remove row t1 without affecting the effect of mutation testing. After applying objects reduction t3=>t1, the reduced context table of TABLE 2 is shown as TABLE 3.

TABLE 3 : Reduced context table of TABLE 2

	m2	m4	m5	m6
t2		X		
t3	X		X	
t4				X

Owner Reduction rule: For each mutant m_i, remove row of test case t that kills the mutant m_i, remove columns for mutants killed by the test case t.

If the context table is empty after owner reduction, then we can get the minimized test suite T_{min}. For example, the reduced context table of our example, shown as TABLE 3 will become an empty table after owner reduction. Thus, we get the minimized test suite {t2, t3, t4}.

Related works

Test suite reduction lower costs by reducing test suite to a minimal subset and satisfy equivalent coverage of the original test suite for a specified test criterion[13]. Formal concept analysis (FCA) is a hierarchical clustering technique[3] for objects with discrete attributes. There are some works to study about test suite reduction based on FCA.

Sampath etc[8]. presented an algorithm based on concept analysis for reducing a test suite for web applications. In their work, one test case in the concept lattice is selected to generate a reduced test suite to cover all the URLs covered by the unreduced suite. Sriraman etc[7]. presented a delayed-greedy algorithm based on concept analysis to select a minimal cardinality subset of a test suite that covers all the requirements covered by the test suite. In their experiments, their algorithm always selected same size or smaller size test suite than that selected by other heuristics algorithm. This work referred to their work, the big differences are we use concept analysis for mutation testing and proposed a test reduction algorithm. Jiang Yuting etc[13]. proposed a method to select test cases in the process of mutation testing to reduce testing cost. They defined the mutation distance to represent the semantic difference between the original program and the mutated program. And then use it to guide the selection of test cases.

Other technology to reduce the computational cost of mutation testing is optimizing the mutant execution process, such as, strong mutation, week mutation and firm mutation.

CONCLUSIONS AND FUTURE WORKS

Mutation testing is an effective but expensive testing method. Formal concept analysis (FCA) can be used for deriving implicit relationships between objects based on attributes. In order to reduce the cost of mutation testing, we present an algorithm for mutation test generation, and then give some reduction rules to reduce the set of test suite which used to kill mutants based on FCA. Results showed our method can generate smaller size test suite than other methods. We hope this approach can do some help to mutation testing.

Our future works include developing our system prototype and using this method to apply other data clustering problems.

ACKNOWLEDGMENT

This work was supported in part by the National Natural Science Foundation of China (61272036).

REFERENCES

[1] Y.Jia, M.Harman; An analysis and survey of the development of mutation testing, IEEE Transactions on Software Engineering (TSE), **(2010)**.

[2] W.E.Wong, A.P.Mathur; "Reducing the cost of mutation testing: An empirical study," Journal of Systems and Software, December, **31(3)**, 185–196 **(1995)**.

[3] S.Hussain, M.Harman; "Mutation clustering", Master Thesis, King's College London, UK, **(2008)**.

[4] C.Ji, Z.Chen, B.Xu, Z.Zhao; "A novel method of mutation clustering based on domain analysis," in Proceedings of the 21st International Conference on Software Engineering and Knowledge Engineering (SEKE'09), Boston, Massachusetts: Knowledge Systems Institute Graduate School, July, 1-3 **(2009)**.

[5] A.J.Offutt, G.Rothermel, C.Zapf; "An experimental evaluation of selective mutation," In Proceedings of the 15th International Conference on Software Engineering (ICSE'93), Baltimore, Maryland: IEEE Computer Society Press, May, 100–107 **(1993)**.

[6] K.Ayari, S.Bouktif, G.Antoniol; "Automatic mutation test input data generation via ant colony", *GECCO'07*, July, London, England, United Kingdom, **7–11**, 1074–1081 **(2007)**.

[7] Sriraman Tallam, Neelam Gupta; A concept analysis inspired greedy algorithm for test suite minimization, PASTE'05, Lisbon, Portugal, **(2005)**.

[8] S.Sampath, V.Mihaylov, A.Souter, L.Pollock; "A scalable approach to user-session based testing of web applications through concept analysis," In proceedings of automated software engineering, 19th International Conference on (ASE'04) Linz, Austria, September, **(2004)**.

[9] P.R.Nannan He, k.Daniel; Test-case generation for embedded simulink via formal concept analysis, DAC 2011, San Diego, California, USA, June, 5-10, **(2011)**.

[10] A.J.Offutt, A.Lee, G.Rothermel, R.H.Untch, C.Zapf; An experimental determination of sufficient mutation operators, ACM Transactions on Software Engineering and Methodology, **5(2)**, 99–118, April 1996.

[11] A.Siami Namin, J.Andrews; Finding sufficient mutation operators via variable reduction, In Mutation, Raleigh, NC, USA, November 2006, **(2006)**.

[12] Y.S.Ma, J.Offutt, Y.R.Kwon; "MuJava : An automated class mutation system," Software Testing, Verification and Reliability, June, **15(2)**, 97–133 **(2005)**.

[13] Jiang Yuting, Li Bixin; Novel technique for cost reduction in mutation testing, Journal of Southeast University (English Edition), Mar., **27(1)**, 17-21 **(2011)**.