Mutation Selection: Some Could be Better than All

Zhiyi Zhang[1,2], Dongjiang You[1,2], Zhenyu Chen[1,2],
Yuming Zhou[1] and Baowen Xu[1]

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

Abstract. In previous research, many mutation selection techniques have been proposed to reduce the cost of mutation analysis. After a mutant subset is selected, researchers could obtain a test suite which can detect all mutants in the mutant subset. Then they run all mutants over this test suite, and the detection ratio to all mutants is used to evaluate the effectiveness of mutation selection techniques. The higher the ratio is, the better this selection technique is. Obviously, this measurement has a presumption that the set of all mutants is the best to evaluate test cases. However, there is no clearly evidence to support this presumption. So we conducted an experiment to answer the question whether the set of all mutants is the best to evaluate test cases. In this paper, our experiment results show that a subset of mutants may be more similar to faults than all the mutants. Two evaluation metrics were used to measure the similarity – rank and distance. This finding reveals that it may be more appropriate to use a subset rather than all the mutants at hand to evaluate the fault detection capability of test cases.

1 Introduction

Mutation analysis is a fault-based testing technique that has been used to identify or create adequate test cases effectively. It was first proposed by [6] and [3]. In mutation analysis, a faulty program termed mutant is generated by seeding a fault into the original program, and the transformation rule to generate mutants is called mutation operator. Then we can execute a test case on a mutant and the original program to compare their outputs. If their outputs are different, we say that the mutant can be detected by this test case. Otherwise, this mutant survives. However, there are some mutants that cannot be detected by any test case and these mutants are deemed to be equivalent mutants. Automatically detecting equivalent mutants has been proved to be an undecidable problem [10].

Although mutation analysis is an effective evaluation for testing experiment [1], it has seldom been used in practice because of its high cost. Even a small program can generate numerous mutants. Obviously, compiling and executing such numerous mutants are heavy burdens in mutation analysis. In order to alleviate these burdens, many cost reduction techniques have been developed. A natural idea is to select a subset from all the mutants, while maintaining the effectiveness to evaluate the fault detection capability of test cases. Operator-based selection, clustering-based selection,

Zhang Z., You D., Chen Z., Zhou Y. and Xu B..
Mutation Selection: Some Could be Better than All.
DOI: 10.5220/0003559700100017
In *Proceeding of the 1st International Workshop on Evidential Assessment of Software Technologies* (EAST-2011), pages 10-17
ISBN: 978-989-8425-58-4

and random selection are three representative mutation selection techniques.

Previous research showed that mutants can be used to replace faults to evaluate the fault detection capability of test cases [1]. Most of the existing efforts have an implication that using all mutants to evaluate the fault detection capability of test cases will increase the validity of their conclusions than using a subset of mutants. After selecting a subset of mutants, we can obtain test cases that can detect all the mutants in this subset. Then we execute all non-equivalent mutants on these test cases and calculate the mutation detection ratio. The higher the ratio is, the better the subset is. However, there is no convincing evidence whether this methodology is trustworthy because some mutants may have negative affection in mutation analysis.

In this paper, we investigated the similarity between faults and all mutants generated using mutation operators, and between faults and subsets of mutants. Our experiment results show that a subset of mutants is more similar to faults in the ability of evaluating test cases. This finding reveals that it may be more appropriate to use a subset rather than all the mutants at hand to evaluate the fault detection capability of test cases in some studies.

The rest of this paper is organized as follows. Section 2 introduces our experiment. Section 3 reports the results and analysis. Section 4 presents the conclusions and future work.

2 Experiment

We are interested in the following research question:

When considering the ability of evaluating test cases, can a subset of mutants be better than all mutants? i.e., can a subset of mutants be more similar to faults than all mutants?

2.1 Subject Programs

We used seven Siemens C programs as subjects [5], [11]. We obtained all the subject programs and the corresponding test cases from *Software-artifact Infrastructure Repository (SIR)* [4]. Table 1 shows the basic information of the seven programs.

Table 1. Subject Programs.

Programs	Lines of Code	Test Pool Size	Non-Equivalent Faults	Non-Equivalent Mutants
tcas	137	1608	40	4011
tot_info	281	1052	23	7870
schedule	296	2650	9	3681
schedule2	263	2710	9	4862
print_tokens	343	4130	7	9165
print_tokens2	355	4115	10	8765
replace	513	5542	31	19861

2.2 Techniques for Generating Mutants

In our experiment, we used the following four techniques to obtain a set of mutants.

For all mutants, we used a tool called ProteumIM2.0 [2], which implemented 108 mutation operators for C language. For each subject program, we generated all mutants with all the mutation operators, and equivalent mutants are discarded in our experiment.

In order to increase the validity of our experiment, we generated the subsets of mutants with the same size (denoted as *n*) when using Offutt et al. operator-based selection, clustering-based selection and random selection.

For operator-based selection, we preferred several operators to all operators, and we generated the operator subset with Offutt et al. 5 operators (ABS, UOI, LCR, AOR and ROR) [9], [8].

For clustering-based selection [7], we classified all non-equivalent mutants into several different clusters using simple K-means based on the detectable test cases. According to the characteristics of clustering, mutants in the same cluster can always be detected by similar test cases. Then we can select several mutants from one cluster to represent all the mutants in that cluster. In our experiment, the number of clusters is *20*, and we randomly selected *n/20* mutants from each cluster. If the number of mutants in a cluster was less than *n/20*, we selected all mutants in this cluster.

For random selection, we randomly selected *n* mutants from all non-equivalent mutants to obtain the subset of mutants.

2.3 Generate Test Suite

We generated a test suite of 512 test cases (the maximum number of test cases allowed by the tool we used) from the test pool by the following procedure:

We used the *test mode* in Proteumim2.0. In this mode, if a mutant has been detected by a test case, this mutant will not be executed by the remaining test cases, and if a test case cannot detect any survived mutant, this test case is defined as redundant test case and thus can be removed from the test suite.

We randomly selected 512 test cases to form a test suite. After running this test suite, we removed redundant test cases (assume the number is *m*) from the test suite and added another *m* test cases from the remaining test cases in the test pool. We repeated this process until all the test cases in the test pool had been executed (*m* was always greater than zero in our experiment). As such, the mutant that can be detected by the test pool can also be detected by this test suite. If the number of test cases in this test suite was less than 512 (all of our subject programs met this condition), we randomly selected test cases that were not in this test suite from the test pool to form a test suite (denoted as *TS*) with exactly 512 test cases.

2.4 Generate Results

We randomly selected test cases from *TS* using a selection rate x% from 25% to 100% in steps of 5%. We denoted these subsets of *TS* as TS_i (*i* is from 1 to 16). Then we ran faulty versions on TS_i and recorded the number of faults that can be detected by TS_i. Thus we could obtain the fault detection ratio as follows:

KF_i = (the number of faults that can be detected by TS_i) / (the number of all faults)

For the sets of mutants generated using each of the four selection techniques, we define mutation detection ratio of this set as follows:

KT_i = (the number of non-equivalent mutants that can be detected by TS_i in mutant set) / (the number of non-equivalent mutants in mutant set)

In order to increase the validity of our experiment, we repeated our experiment five times for each TS_i, and computed the average over them as the final result.

3 Results and Analysis

In this section, we firstly present our experiment results and the process of evaluating the results, and answer the research question we proposed in Section 2.1. Then we analyze the possible reasons that could explain the results of our experiment.

As previously mentioned in this paper, the more similar the ability of evaluating test cases of a mutant set is to faults, the better this mutant set is. So we use the following two evaluation metrics to measure the similarity between mutant sets and faults.

3.1 Evaluation by Rank

Table 2. Evaluation by Rank on *tcas*.

Selection Rate	All	Operator	Clustering	Random
25%	3	1	4	2
30%	3	1	4	2
35%	3	1	4	2
40%	3	1	4	2
45%	3	1	4	2
50%	2	1	4	3
55%	3	1	4	2
60%	3	1	4	2
65%	2	1	4	3
70%	3	1	4	2
75%	3	1	4	2
80%	2	1	4	3
85%	2	1	4	3
90%	2	1	4	3
95%	2	4	1	3
100%	1	1	1	1
Sum	40	19	58	37

We firstly calculated the numerical difference between KF_i and KT_i. Then we compared the values of differences among all mutants and subsets of mutants. We assigned the smallest difference as *1*, the second smallest difference as 2, etc. For each technique, there will be 16 such values of differences. Table 2 shows those values on *tcas*. The column indicates the four techniques, the row indicates selection rate x% from 25% to 100% in steps of 5%, in a total of 16 rows.

We summed up these 16 values as the rank value. The lower the rank value is, the better this technique is. Table 3 shows our experiment results. The column indicates the four techniques, the row indicates different subject programs, and the number is the rank value.

Table 3. Evaluation by Rank.

Programs	All	Operator	Clustering	Random
tcas	40	**19**	58	37
tot_info	38	**26**	45	47
schedule	40	**28**	52	37
schedule2	39	37	**36**	40
print_tokens	35	38	47	**34**
print_tokens2	44	36	**28**	49
replace	36	**26**	52	42

3.2 Evaluation by Distance

We calculated the Euclidean distance between the result sequences of fault detection ratios (KF_1 to KF_{16}) and that of mutant detection ratios (KT_1 to KT_{16}) for each technique and each subject program. The lower the distance is, the better this technique is. We summarized our results in Table 3. The column indicates the four techniques, the row indicates different subject programs, and the number is the Euclidean distance.

$$\text{Euclidean distance} = \sqrt{\sum_{i=0}^{16} (KF_i - KT_i)^2}$$

Table 4. Evaluation by Distance.

Programs	All	Operator	Clustering	Random
tcas	0.4825	**0.4201**	0.5205	0.4694
tot_info	0.2229	**0.2194**	0.2214	0.2232
schedule	0.9071	**0.8405**	0.9163	0.9117
schedule2	0.3108	**0.2739**	0.3237	0.3237
print_tokens	0.4129	0.4283	0.4435	**0.4014**
print_tokens2	0.1826	0.1897	0.2000	**0.1777**
replace	0.1608	**0.1597**	0.1993	0.1751

3.3 Result Analysis

To sum up, there is always a subset of mutants which is better than all mutants. And in none of the seven subject programs, all mutants can be the best. It also means that the subset selected by three selection techniques, especially by Offutt et al. operators has a better ability for evaluating test cases than all mutants. We provide the following explanations for this phenomenon in our experiment:

Firstly, the faults in Siemens programs are more difficult to be detected than most mutants, and this fact has been confirmed in [1]. The faults that can be detected by 350 or more test cases have been discarded from the subject programs. Therefore, the remaining faults are difficult to be detected.

Secondly, the mutants in operator subset are much more difficult to be detected than most of other mutants. Taking ABS for example, detecting the mutants generated by ABS requires the test cases which select from different parts of the input domain related to the mutated expression [12]. Therefore, only a small number of test cases can detect these mutants.

Thirdly, because of the characteristics of clusters, the mutants in the same cluster can be detected by the similar test cases. Therefore, if the test cases we selected can detect one mutant, the mutants we selected from the same cluster may also be detected by those test cases. Consequently, most mutants in clustering subsets can be detected.

Finally, there are two kinds of mutation operators in proteumIM2.0, unit operators and interface operators. The unit operators include modifications to operands, statements and operators in expressions. The interface operators are related to modifications to methods and classes features. In our experiment, however, the mutants in operator subset were all generated by unit operators, others subsets include mutants which were generated by interface operators. We suppose that interface operator mutants are easier to be detected than unit operator mutants. So for many programs in our experiment, the test sets detected fewer mutants in operator subset than other subset.

3.4 Threats to Validity

Threats to internal validity are uncontrolled factors that are also responsible for our results. The main threat is that there are defects in the process of generating, selecting and clustering mutants. To reduce this threat, we used ProteumIM2.0 to generate mutants and Offutt subset, and used Weka to cluster mutants with simple K-means. ProteumIM2.0 and Weka are tools that have been widely applied in previous research works.

Threats to external validity are the representativeness of our subject programs and experiment procedures. The main threat is that the subject programs with their test cases and faulty versions may not have generality. To reduce this threat, we chose seven widely used Siemens programs, which contain plenty of test cases and different numbers of faults.

Threats to construct validity are the measurements we used to evaluate our results. To reduce the threat, we applied rank sum measurement and Euclidean distance measurement, which are well-known metrics to measure the (dis)similarity between two sequences.

4 Conclusions and Future Work

In this paper, we conducted an experiment to investigate an important question in mutation analysis. We compared the similarities between faults and all mutants, and between faults and subsets of mutants. We confirmed that using a subset of mutants is more appropriate than using all mutants when evaluating test cases.

In the future work, we plan to do the following research. Firstly, we will conduct experiments on larger programs with more faulty versions. Secondly, we will apply more selection techniques to extend our research. For example, in operator-based selection, we will use Barbosa's operators and Siami Namin's operators, and use other algorithms in clustering-based selection. Thirdly, we will do more research to deeply explain the phenomenon in detail that using the subset of mutants is better than using all mutants when evaluating test cases. Finally, we will figure out which selection technique should we use when we want to select a subset of mutants from all mutants. We propose to apply more programs with different sizes and structures to answer this question.

Acknowledgements

The work described in this article was partially supported by the National Natural Science Foundation of China (90818027, 60803007, 60803008, 61003024), the Fundamental Research Funds for the Central Universities.

References

1. Andrews, J. H., Briand, L. C., and Labiche, Y. (2005). Is mutation an appropriate tool for testing experiments? ICSE 2005: 402-411.
2. Delamaro, M. E. and Maldonado, J. C. (1996) Proteum – A Tool for the Assessment of Test Adequacy for C Programs – User's guide.
3. DeMillo, R. A., Lipton, R. J., and Sayward, F. G. (1978). Hints on Test Data Selection: Help for the Practicing Programmer. Computer 11(4): 34–41 (1978).
4. Do, H., Elbaum, S. G., Rothermel, G. (2005). Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact. Empirical Software Engineering 10(4): 405-435 (2005).
5. Elbaum, S. G., Malishevsky, A. G., and Rothermel, G., (2000). Prioritizing test cases for regression testing. ISSTA 2000: 102-112.
6. Hamlet, R. G., (1977). Testing Programs with the Aid of a Compiler. IEEE Transactions on Software Engineering 3(4): 279-290 (1977).
7. Hussain, S., (2008). Mutation Clustering. Master's Thesis, King's College London, Strand, London, 2008.
8. Mathur, A. P., (1991). Performance, Effectiveness, and Reliability Issues in Software Testing. COMPSAC 1991: 604–605.
9. Offutt, A. J., Lee, A., Rothermel, G., Untch, R. H., and Zapf, C., (1996). An Experimental Determination of Sufficient Mutant Operators. ACM Transactions on Software Engineering and Methodology 5(2): 99-118 (1996).

10. Offutt, A. J. and Pan, J. (1977). Automatically Detecting Equivalent Mutants and Infeasible Paths. Software Testing, Verification and Reliability 7(3): 165-192 (1997).
11. Rothermel, G., Harrold, M. J., Ostrin, J., and Hong, C., (1998). An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites. ICSM 1998: 34-43.
12. Wong, W. E. and Mathur, A. P., (1995). Reducing the cost of mutation testing: An empirical study. Journal of Systems and Software 31(3): 185-196 (1995).