Automatic Mutation based Test Data Generation

Mike Papadakis
Department of Informatics
Athens University of Economics and Business
mpapad@aueb.gr

Nicos Malevris
Department of Informatics
Athens University of Economics and Business
ngm@aueb.gr

ABSTRACT
This paper proposes a search based test data generation approach for mutation testing. The proposed approach uses a novel dynamic execution scheme in order to both introduce mutants and to effectively guide the search process towards generating test cases able to kill those mutants. A novel fitness function and its integration with a dynamically adjusted mechanism are also proposed. Preliminary experimentation with a hill climbing search based approach reveals its power when compared against a previously proposed one and random testing.

Categories and Subject Descriptors
D.2.5 [Testing and Debugging]: Testing tools

General Terms
Algorithms, Experimentation

Keywords
Test input data generation, Search based testing, Mutation testing

1. INTRODUCTION
Mutation testing or mutation analysis is a well known fault-based criterion. Mutation analysis introduces purposely built errors into programs under test by making alterations (mutants) to the tested code based on a set of simple syntactic rules called mutant operators. The purpose of injecting faults into programs is to both guide the generation of test cases to reveal them on the one hand and to assess the test data quality on the other. To this extent, testing seeks to reveal the mutants, which when detected are termed "killed" and "live" in the opposite case. Testing adequacy is measured using the mutation score, defined as the ratio of the number of the killed mutants to the entire number of candidate mutants reduced by the number of equivalent ones. Equivalent mutants are those mutants that can not be killed by any test case.

The practical use of an adequacy criterion requires the automated generation of test cases according to its requirements. Recently, search based optimization techniques and tools have succeeded in automating the test case generation activity quite effectively. This paper introduces an automated framework that produces test cases based on mutation testing. In the proposed framework, the mutants are automatically generated based on a novel version of the mutant schemata technique similar to [7] and [8] for performing both mutation and search based testing. The use of mutant schemata for mutation test data generation purposes has also been investigated in the past, in the context of weak mutation [8], utilizing existing structural testing tools, and in the context of strong mutation using dynamic symbolic execution [7]. Here the proposed approach incorporates a hill climbing algorithm known as the alternating variable method (AVM) proposed by Korel [5] for searching and producing the sought test data.

The origins of the present approach are due to the utilized dynamic fitness scheme and evaluation. Thus, it becomes possible to effectively direct the search process towards killing the targeted mutants. A performed case study suggests that it can be more effective than random testing and a previously proposed approach [1]. This comparison also reveals the inefficiency of the previously proposed approach [1] to practically guide the search towards killing mutants when equivalent ones are also included.

GECCO'11, July 12–16, 2011, Dublin, Ireland.
ACM 978-1-4503-0690-4/11/07.

2. FITNESS FUNCTION
The present framework utilizes a fitness function composed of four parts. The first two are known as the "approach level" and the "branch distance" introduced by Wegener et al. [9] in the context of structural testing, the third one is named "mutation distance" and the fourth one is named "impact distance".

Mutation distance as introduced in this paper reflects the branch distance measure on mutants. This approach is in line with the suggestions made by Bottaci [2] for the mutation testing fitness calculations. It should be noted that these three measures guide the search towards fulfilling the reachability and mutant necessity constraints proposed by Demillo and Offutt [3]. Mutation distance fitness calculations should quantify the distance of changing the mutant and original program predicates outcome (at the mutated statement) in order to be effective [3]. To achieve this, the following expression, where O and M denotes the original and the mutant predicates fitness calculations is used.

$$pmd = min[Tfit(O) + Ffit(M), Tfit(M) + Ffit(O)]$$

Impact distance tries to approximate the mutant sufficiency condition [3]. To achieve this, it is suggested to record the impact (differences on the execution paths between the original and mutant program versions) [4] of each mutant during the test generation process. For each program node that has been impacted the ratio of the killed over the total number of mutants is recorded. Informally, as tests are produced and executed with mutants the nodes are ranked according to their ability to expose mutants, when they are impacted. Impact distance reflects the approach level and the branch distance on the mutant program towards covering a selected top ranked node.

Table 1. Proposed fitness

fitness *= reach dis + mutation dis + impact dis*
reach dis *= 2 * approach level + normalized(branch distance)*
mutation dis *= normalized(mutation distance) + normalized(pmd)*
impact dis *= approach level + normalized(branch distance)*

Table 2. Mutants killed by the utilized fitness functions per test subject

Test Subjects	Mutants num	Random	Reach	Infect	Impact	DReach	DInfect	DImpact
Triangle	*166*	*102.2*	*94*	*103*	*103.4*	*96.4*	*103*	*103.2*
Trityp	*349*	*125.6*	*173.8*	*178.4*	*184.8*	*205.4*	*210.4*	*223*
Triangle	*421*	*102*	*131*	*144.4*	*146.2*	*143.8*	*148.6*	*185*
Remainder	*324*	*205.8*	*201.4*	*206*	*206*	*201.4*	*206*	*206*
Callendar	*327*	*189*	*165*	*195.2*	*193.2*	*168.6*	*198.8*	*200*
Cancel	*866*	*712.6*	*686.2*	*732.2*	*732.6*	*709.26*	*732*	*733.2*
FourBalls	*225*	*187.2*	*183.2*	*185*	*186.8*	*181*	*185.8*	*188*
Quadratic	*81*	*59.07*	*58*	*61.22*	*61.8*	*58*	*60.6*	*63*

Conclusively, the proposed fitness function guides the search towards reaching (approach level + branch distance) a mutant, causing a discrepancy at the mutation point (mutation distance), propagates it to the outcome of the mutant statement (predicate mutation distance) and impacts specific likely to expose mutants, program nodes (here referred to as impact nodes). Computation of the overall fitness of the test cases is performed based on the equation presented in Table 1.

3. DYNAMIC APPROACH LEVEL

Mutation testing introduces a vast number of mutants which are spanned across the whole program structure. It was observed that trying to kill them, results into covering - reaching many other mutants (possible hard to reach) collaterally i.e. without aiming at them. It is noted that many mutants are equivalent and thus by their definition aiming at them will result in a waste of effort. In practice, these two characteristics of mutation can provide useful information to assist the killing of some other mutants. Such an approach is the dynamic approach level as proposed in this paper.

The rationale behind the use of the standard approach level [9] is to include only the structural elements (control dependencies) that must be traversed by any of the possible sought test cases. The dynamic approach level extends this rationale by trying to consider both the necessary control and data dependencies at the same time. Consider a case where in order to traverse a targeted branch requires the program execution to execute a specific program statement (data dependency) that is not part of the control dependencies of the targeted branch. Then, all the possible test cases that traverse this branch also traverse the specific program statement. The dynamic approach level identifies all the common structural elements that traverse the produced test cases and thus, necessary data dependencies too. This way the feasible path information gained during the whole search process can be reused for infecting and eventually killing the introduced mutants.

The dynamic approach level is defined as the intersection of all the nodes that are contained in all the encountered execution paths that reach a targeted node. Thus, for example if a target node is x and during the search process y different execution paths have been encountered that lead to node x, then the dynamic approach level is formed as the common nodes of these y paths. If there is not any path leading to node x then the standard approach level is used. This approach relies on the excessive search performed.

4. EXPERIMENTS

The conducted experiment includes the production of test cases based on random testing and three fitness functions with the use of the AVM method. The first fitness function named "Reach" uses only the reach distance of the proposed fitness (Table 1). The second one named "Infect" uses the reach and mutation distances and the third one named "Impact" utilizes the whole fitness. The same (3 variants) fitness functions where used utilizing the dynamic approach level and named as "DReach", "DInfect" and "DImpact" respectively. This utilization was performed by iteratively performing attempts to kill mutants, once based on the standard approach level and once based on the dynamic approach level. For each program the test generation process considered up to 50,000 fitness evaluations per introduced mutant or random test generations (for random testing). This is a considerably high number of evaluations but it is forced due to the existence of equivalent mutants. The results reported in Table 2 are based on the mutant operator set propose in [6] and record the average obtained values by 10 independent application times.

5. CONCLUSION AND FUTURE WORK

This paper proposes a practical approach to produce mutation based test data. Preliminary results suggest that the proposed fitness functions can outperform a previously proposed one and random testing as well. Also the use of the dynamic approach level can increase the effectiveness of the approach. In future, the inclusion of other search based approaches such as evolutionary testing are planned. Further investigation is needed in order to determine the benefits of the dynamically adopted approach level and its optimal use in search based testing.

6. REFERENCES

[1] Ayari, K., Bouktif, S. and Antoniol, G. Automatic mutation test input data generation via ant colony. In Proc. of the 9th annual conference on Genetic and evolutionary computation, pages 1074-1081, 2007.

[2] Bottaci, L. A genetic algorithm fitness function for mutation testing. In SEMINAL: Software Engineering using Metaheuristic INovative Algorithms, Workshop 8, ICSE 2001, pages 3-7, 2001.

[3] DeMillo, R. A. and Offutt, A. J. Constraint-Based Automatic Test Data Generation. IEEE Trans. Softw. Eng., 17, 9 (1991), 900-910.

[4] Fraser, G. and Zeller, A. Mutation-driven generation of unit tests and oracles. In Proc. of the 19th international symposium on Software testing and analysis, pages 147-158, 2010.

[5] Korel, B. Automated Software Test Data Generation. IEEE Trans. Softw. Eng., 16, 8 (1990), 870-879.

[6] Offutt, A. J., Lee, A., Rothermel, G., Untch, R. H. and Zapf, C. An experimental determination of sufficient mutant operators. ACM Trans. Softw. Eng. Methodol., 5, 2 (1996), 99-118.

[7] Papadakis, M. and Malevris, N. Automatic Mutation Test Case Generation Via Dynamic Symbolic Execution. In ISSRE, 2010.

[8] Papadakis, M., Malevris, N. and Kallia, M. Towards automating the generation of mutation tests. In Proc. of the 5th Workshop on Automation of Software Test, pages 111-118, 2010.

[9] Wegener, J., Baresel, A. and Sthamer, H. Evolutionary test environment for automatic structural testing. Information and Software Technology, 43, 14 (2001), 841-854.