On the Utility of Dominator Mutants for Mutation Testing

Bob Kurtz
Software Engineering
George Mason University
Fairfax VA, USA
rkurtz2@gmu.edu

ABSTRACT

Mutation testing has been shown to support the generation of test sets that are highly effective at detecting faults. However, practitioner adoption of mutation testing has been minimal in part because of problems that arise from the huge numbers of redundant and equivalent mutants that are generated. The research described here examines the relationship between mutants and attempts to reduce the number of redundant and equivalent mutants in order to make mutation testing more practical for the software tester.

CCS Concepts

•Software and its engineering → Software testing and debugging;

Keywords

Mutation analysis, subsumption, dominator mutants

1. RESEARCH PROBLEM

Mutation testing [7] is a test criterion that generates a set of alternate programs, called *mutants*, and then challenges the tester to design tests to detect the mutants. Tests that cause a mutant to behave differently from the original program are said to *kill* the mutant. Some mutants behave exactly the same as the original on all inputs. These are called *equivalent mutants* and cannot be killed. Mutants are generated by mutation operators, rules that generate variants of a given program based on particular syntactic elements.

Early on, researchers observed that mutation operators produced far more mutants than necessary [18]. These *redundant mutants* are mutants that can be killed, but do not improve the test suite by requiring new or different tests. Consequently, these mutants simply waste the tester's time.

Minimal mutation [2] precisely defines redundancy among mutants by identifying *dominator mutants*, which *subsume* other mutants. A test set that kills the dominator mutants will kill all other non-equivalent mutants. Subsumption relationships identify mutant redundancy and can be used to optimize both mutant and test generation. This research seeks to apply the concepts of subsumption and dominator mutants to improve the efficiency of mutation testing.

FSE'16, November 13–18, 2016, Seattle, WA, USA
ACM. 978-1-4503-4218-6/16/11...$15.00
http://dx.doi.org/10.1145/2950290.2983950

2. RELATED WORK

The large number of mutants generated by mutation testing has long been a recognized problem. Mathur [15] introduced the idea of *constrained mutation* to reduce the number of mutation operators to create fewer mutants. Offutt *et al.* [18] took an empirical approach to defining a subset of selective mutation operators that achieved a mutation score of 0.99 or higher. Wong *et al.* [22] evaluated combinations of mutation operators for efficiency and effectiveness.

Barbosa *et al.* [3] generated a set of mutation operators that reduced the computational cost of mutation testing without losing effectiveness. Namin *et al.* [17] analyzed the Siemens suite programs to identify three high-performing operator sets. Delamaro *et al.* [6] defined a growth model for mutation operator selection, adding operators until a mutation score of 1.00 was achieved. All concluded that there is no single best set of operators for all programs.

Taking mutation operator reduction to an extreme, Untch [21] evaluated the statement deletion (SDL) operator on its own and found it to be competitive with the operator sets found by Namin. Deng *et al.* [8] found that SDL achieved a mutation score close to that of Offutt's *E-selective* operators while generating approximately 80% fewer mutants. Delamaro *et al.* [5] evaluated SDL against programs written in C using Proteum and confirmed Deng's findings.

Other researchers have examined whether selective mutation is more effective than random sampling of similar numbers of mutants. Acree [1] and Budd [4] separately concluded that executing tests that kill a randomly-selected 10% of mutants could provide results close to executing tests that kill the full set of mutants. Wong and Mathur [23] demonstrated similar results. More recently, Zhang *et al.* [26] also found no appreciable difference in performance between selective mutation and random selection. Gopinath *et al.* [9] expanded this investigation using a much larger body of open-source code, again finding that random selection performs as well as any other strategy. Zhang *et al.* [25] combined selective mutation with random mutant selection to further reduce the effort required to achieve a high mutation score.

Several efforts have tried to formalize the widely-held notion that some mutants are more valuable than others with respect to forcing particular test cases. Yao *et al.* [24] suggested *stubborn* mutants that are not killed by a branch-adequate test set. Namin *et al.* [16] hypothesized the ex-

istence of "super mutants" that are difficult to kill and for which a killing test also kills a number of other mutants. Papadakis *et al.* [19] confirmed Ammann's inflation of mutation score, and determined that 68% of recent research papers are vulnerable to threats to validity due to the effect of redundant mutants. The subsumption relation has been studied in a variety of contexts for many years. Jia and Harman defined the notion of subsuming Higher Order Mutants (HOMs) [10], where two or more mutation operations are applied to create a mutant that is harder to kill than the first order mutants from which it is constructed. Ammann *et al.* [2] introduced a formalized description of mutant subsumption, including dynamic mutation subsumption based on the score function as a binary matrix. The thesis described here is based primarily on Ammann's work.

3. PROPOSED APPROACH AND EXPECTED CONTRIBUTIONS

This research seeks to apply the concepts of mutant subsumption and dominator mutants to improve the process of mutation testing by determining approaches to preferentially produce useful mutants while reducing the production of redundant and equivalent mutants.

3.1 Research Results So Far

"Mutant subsumption graphs" [11] refined minimal mutation and defined *true subsumption* and its *static* and *dynamic* approximations. We defined a dynamic approximation of true subsumption and defined the *mutant subsumption graph* (MSG), a graphical notation for representing the subsumption relationship between mutants.

"Static analysis of mutant subsumption" [12] explored static subsumption using symbolic execution on the same small program. We described an algorithm for using *directed incremental symbolic execution* (DiSE) [20] to perform subsumption analysis, and concluded that the limitations of symbolic execution made it impractical to directly determine subsumption relationships. However, symbolic execution proved effective at generating test cases dynamically eliciting an approximation of the subsumption relationship.

"Are we there yet? How redundant and equivalent mutants affect determination of test completeness" [14] formalized decoupled metrics for measuring the number of redundant and equivalent mutants. We showed how redundancy makes it difficult to determine the completeness of a test suite because mutation score quickly increases to nearly 1.0 and then increases very slowly until testing is complete. *Dominator mutation score*, the number of dominator mutants killed divided by the total number of dominator mutants, is a more linear predictor of completeness. Unfortunately dominator score cannot be determined until testing is complete, while mutation score can at least be reasonably estimated at any point during testing.

We then examined how redundancy and equivalency affect the ability of the tester to predict dominator score (and thus test completeness) based on mutation score, and discovered that redundant mutants strongly degrade this ability. Equivalent mutants, perhaps surprisingly, have little effect on the ability to predict completeness. We also examined how redundancy and equivalency affect the amount of work required by the tester, and found that redundancy had a small impact on work because killing any randomly-selected mutant tends to kill many other mutants. Equivalency had a strong impact on work because every equivalent mutant must be individually examined.

"Analyzing the validity of selective mutation with dominator mutants" [13] considered the effectiveness of selective mutation as a function of dominator score per unit work (defined as the number of mutants examined by the tester) based on a small set of sample programs. We found that the traditional *E-selective* mutation operators [18] generate high mutation scores but relatively low dominator scores.

We determined that while various combinations of mutation operators achieved high dominator scores with low work for any particular program in our sample set, there were no such combinations that performed well for *all* of our sample programs. We concluded from this experiment that when viewed through the lens of dominator mutation score, selective mutation as it has been practiced is not especially optimal, and that new approaches are needed in order to generate high dominator scores with reasonable work.

3.2 Future Research Plans

In our future work we seek to develop an approach to customize the generation of mutants based on program structure. To this end, we hope to characterize the program features at mutant injection points and train a machine learning algorithm on a portion of the Defects4J programs to associate program features with the mutation operators that tend to generate dominator or near-dominator mutants at those injection points. Identification of the program feature set is key to success; currently we are considering the mutated statement, the applied mutation operator, the surrounding context (loops, if statements, return statements, etc.) in the control flow graph.

If we can develop such a correlation between program features and effective mutation operators, we could then evaluate the effectiveness of the algorithm by applying it to the remaining Defects4J programs. Evaluation of success would be based on achieving a significantly higher dominator score per unit work as compared to existing techniques like selective mutation or random mutant selection.

Unkilled mutants are a particular concern. An unkilled mutant might truly be equivalent, in which case we want to avoid selecting it. However, it may simply be a mutant that hasn't yet been killed by a test, which suggests that it might actually be a dominator or near-dominator mutant. We conjecture that the machine learning algorithm will reduce these false negatives because they are likely to have similar features to other dominator mutants.

If successful, my research will improve mutation testing by preferentially generating dominator mutants, thus reducing the number of redundant and equivalent mutants. This will improve the ability of the tester to determine how much testing work has been completed and how much remains, while also significantly reducing the amount of work required.

4. ACKNOWLEDGEMENTS

Thank you to Dr. Paul Ammann for his guidance in my research and to my co-authors for their contributions.

5. REFERENCES

[1] A. T. Acree. *On Mutation*. PhD thesis, Georgia Institute of Technology, Atlanta, Georgia, USA, 1980.

[2] Paul Ammann, Marcio E. Delamaro, and Jeff Offutt. Establishing theoretical minimal sets of mutants. In *7th IEEE International Conference on Software Testing, Verification and Validation (ICST 2014)*, pages 21–31, Cleveland, Ohio, USA, March 2014.

[3] E. F. Barbosa, J. C. Maldonado, and A. M. R. Vincenzi. Toward the determination of sufficient mutant operators for C. *Software Testing, Verification, and Reliability, Wiley*, 11(2):113–136, June 2001.

[4] T. A. Budd. *Mutation Analysis of Program Test Data*. PhD thesis, Yale University, New Haven, Connecticut, USA, 1980.

[5] Marcio E. Delamaro, Lin Deng, Serapilha Dureli, Nan Li, and Jeff Offutt. Experimental evaluation of SDL and one-op mutation for C. In *7th IEEE International Conference on Software Testing, Verification and Validation (ICST 2014)*, Cleveland, Ohio, March 2014.

[6] Márcio E. Delamaro, Lin Deng, Nan Li, and Vinicius H. S. Durelli. Growing a reduced set of mutation operators. In *Proceedings of the 2014 Brazilian Symposium on Software Engineering (SBES)*, pages 81–90, Maceió, Alagoas, Brazil, September-October 2014.

[7] Richard A. DeMillo, Richard J. Lipton, and Fred G. Sayward. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, 11(4):34–41, April 1978.

[8] Lin Deng, Jeff Offutt, and Nan Li. Empirical evaluation of the statement deletion mutation operator. In *6th IEEE International Conference on Software Testing, Verification and Validation (ICST 2013)*, pages 65–74, Luxembourg, March 2013.

[9] Rahul Gopinath, Amin Alipour, Iftekhar Ahmed, Carlos Jensen, and Alex Groce. Do mutation reduction strategies matter? Technical report, School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, Oregon, USA, August 2015.

[10] Yue Jia and Mark Harman. Constructing subtle faults using higher order mutation testing. In *2008 Eighth IEEE International Working Conference on Source Code Analysis and Manipulation*, pages 249–258, Beijing, September 2008.

[11] Bob Kurtz, Paul Ammann, Márcio E. Delamaro, Jeff Offutt, and Lin Deng. Mutation subsumption graphs. In *Tenth IEEE Workshop on Mutation Analysis (Mutation 2014)*, pages 176–185, Cleveland, Ohio, USA, March 2014.

[12] Bob Kurtz, Paul Ammann, and Jeff Offutt. Static analysis of mutant subsumption. In *Eleventh IEEE Workshop on Mutation Analysis (Mutation 2015)*, Graz, Austria, April 2015.

[13] Bob Kurtz, Paul Ammann, Jeff Offutt, Márcio E. Delamaro, Mariet Kurtz, and Nida Gökçe. Analyzing the validity of selective mutation with dominator mutants. In *FSE 2016, Proceedings of the ACM SIGSOFT 24nd Symposium on the Foundations of Software Engineering*, page to appear, Seattle, Washington, USA, November 2016, to appear.

[14] Bob Kurtz, Paul Ammann, Jeff Offutt, and Mariet Kurtz. Are we there yet? How redundant and equivalent mutants affect determination of test completeness. In *Twelfth IEEE Workshop on Mutation Analysis (Mutation 2016)*, Chicago, Illinois, USA, April 2016.

[15] Aditya Mathur. Performance, effectiveness, and reliability issues in software testing. In *Proceedings of the Fifteenth Annual International Computer Software and Applications Conference*, pages 604–605, Tokyo, Japan, September 1991.

[16] Akbar Namin, Xiaozhen Xue, Omar Rosas, and Pankaj Sharma. Muranker: A mutant ranking tool. *Software Testing, Verification and Reliability*, to appear. Published online August 2014.

[17] Akbar Siami Namin, James H. Andrews, and Duncan J. Murdoch. Sufficient mutation operators for measuring test effectiveness. In *Proceedings of the 30th International Conference on Software Engineering*, pages 351–360, Chicago, Illinois, USA, 2008. ACM.

[18] Jeff Offutt, Ammei Lee, Gregg Rothermel, Roland Untch, and Christian Zapf. An experimental determination of sufficient mutation operators. *ACM Transactions on Software Engineering Methodology*, 5(2):99–118, April 1996.

[19] Mike Papadakis, Christopher Henard, Mark Harman, Yue Jia, and Yves Le Traon. Threats to the validity of mutation-based test assessment. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'16)*, page to appear, Saarbrücken, Germany, July 2016.

[20] S. Person, G. Yang, N. Rungta, and S. Khurshid. Directed incremental symbolic execution. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 504–515, San Jose, California, June 2011. ACM SIGPLAN.

[21] Roland Untch. On reduced neighborhood mutation analysis using a single mutagenic operator. In *ACM Southeast Regional Conference*, pages 19–21, Clemson SC, March 2009.

[22] W. Eric Wong and Aditya P. Mathur. Reducing the cost of mutation testing: An empirical study. *Journal of Systems and Software, Elsevier*, 31(3):185–196, December 1995.

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

[24] Xiangjuan Yao, Mark Harman, and Yue Jia. A study of equivalent and stubborn mutation operators using human analysis of equivalence. In *Proceedings of the 36th International Conference on Software Engineering*, pages 919–930, Hyderabad, India, May 2014.

[25] Lingming Zhang, Milos Gligoric, Darko Marinov, and Sarfraz Khurshid. Operator-based and random mutation selection: Better together. In *28th IEEE International Conference on Automated Software Engineering*, pages 92–102, November 2013.

[26] Lu Zhang, Shan-San Hou, Jun-Jue Hu, Tao Xie, and Hong Mei. Is operator-based mutant selection superior to random mutant selection? In *32nd ACM/IEEE International Conference on Software Engineering*, pages 435–444, May 2010.