Moving the Goalposts: Coverage Satisfaction Is Not Enough[*]

Gregory Gay[*], Matt Staats[‡], Michael W. Whalen[*], and Mats P.E. Heimdahl[*]

[*]Department of Computer Science & Engineering
University of Minnesota, USA
greg@greggay.com,
[whalen,heimdahl]@cs.umn.edu

[‡]Interdisciplinary Centre for
Security, Reliability and Trust
University of Luxembourg
matthew.staats@uni.lu

ABSTRACT

Structural coverage criteria have been proposed to measure the adequacy of testing efforts. Indeed, in some domains—e.g., critical systems areas—structural coverage criteria must be satisfied to achieve certification. The advent of powerful search-based test generation tools has given us the ability to generate test inputs to satisfy these structural coverage criteria. While tempting, recent empirical evidence indicates these tools should be used with caution, as merely achieving high structural coverage is not necessarily indicative of high fault detection ability. In this report, we review some of these findings, and offer recommendations on how the strengths of search-based test generation methods can alleviate these issues.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Verification

Keywords

Software Testing, Automated Test Generation, Structural Coverage

1. INTRODUCTION

Central to software test selection are *structural coverage criteria*, which measure test suite adequacy in terms of coverage over the structural elements of the system under test, such as statements or control flow branches. These criteria are so trusted that they are required for certification when testing avionics systems.

These adequacy metrics can be easily adapted as objective functions for search-based optimization algorithms, and as a result, there has been rapid progress in the creation of automated test generation tools that direct the generation process to satisfy structural coverage criteria [1]. Such tools promise to improve coverage and reduce the cost associated with test creation.

[*]This work has been supported by NASA Ames Cooperative Agreement NNA06CB21A, NSF grants CCF-0916583, CNS-0931931, and CNS-1035715, and the Fonds National de la Recherche, Luxembourg (FNR/P10/03).

In principle, this represents a success for software engineering—an arduous engineering task has been automated. However, in actuality, the effectiveness of test suites automatically generated to satisfy various structural coverage criteria has not been firmly established in practice. Of crucial importance is the *method* of test generation: while evidence establishing that coverage is positively correlated with fault detection, such evidence typically holds the method of test generation fixed, e.g., demonstrating that using structural coverage to guide random test generation provides better tests than purely random tests [13]. Consequently, much of the work in test generation (search-based and otherwise) fails to address the effectiveness of the generated tests, and the studies that have examined this issue have returned mixed results [7, 5, 3].

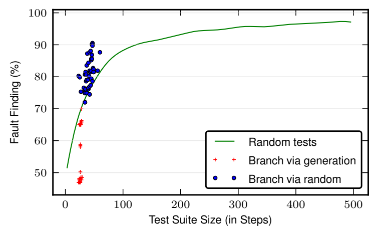
In recent work, we have examined the effectiveness of automated test generation in depth and found that test inputs generated specifically to satisfy structural coverage criteria via counterexample-based test generation were typically *less effective* than randomly generated test inputs [13]. These results are disconcerting—given the central role of coverage criteria in testing research and practice, and the rise of automated tools that can provide such coverage, the temptation exists to automate the entire testing process. Our results indicate that it is not sufficient to simply maximize the level of code coverage when generating tests. Recent research suggests that a number of factors that are currently not well understood can strongly impact the effectiveness of the testing process, such as the oracle used or the structure of the program under test [16].

These findings lead us to the conclusion that code coverage is merely one factor in the broader objective of generating effective test input. Search-based test generation algorithms must evolve to take into account additional factors such as fault propagation, system structure, and the execution points monitored by the test oracle. In this report, we discuss our findings and make recommendations on the next generation of search-based test-generation approaches.

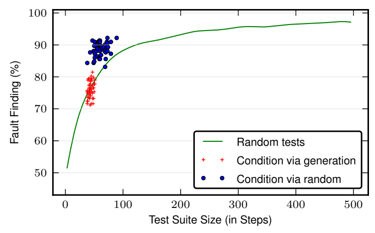
2. THE CENTRAL QUESTION

Advances in search-based test generation should be celebrated; however, we seek evidence that using such generation techniques to maximize structural coverage yields test suites that are more effective at fault detection than naïve generation methods such as random testing. That is, are tests automatically generated to achieve coverage actually useful for finding faults in the system under test?

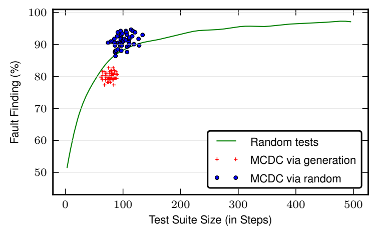
Empirical studies comparing structural coverage criteria have mixed results. Juristo et al. provide a survey of much of the existing work [8]. With respect to branch coverage, they note that some authors (such as Hutchins et al. [7]) find that branch coverage outperforms random testing, while others (such as Frankl and Weiss [5]) discover the opposite. Namin and Andrews have found coverage levels are positively correlated with fault finding effectiveness [11].



(a) Branch



(b) Condition



(c) MC/DC

Figure 1: Percentage of faults identified compared to test suite size for generated test suites, coverage-satisfying subsets of randomly generated tests, and pure random test generation.

Theoretical work comparing the effectiveness of partition testing against random testing yields similarly mixed results. Chen and Yu indicate that partition testing is not necessarily more effective than random testing [3]. Later theoretical work by Gutjahr [6], however, provides a stronger case for partition testing. Kandl and Kirner evaluate MC/DC using an example from the automotive domain and note less than ideal fault finding [9]. Dupuy and Leveson evaluate the MC/DC as a complement to functional testing, finding that the use of MC/DC improves the quality of tests [4]. None of these studies, however, compare the effectiveness of test generation to satisfy MC/DC to other forms of test generation. They, therefore, do not indicate if test suites satisfying MC/DC are truly effective, or if they are effective merely because MC/DC test suites are generally quite large. Arcuri et al. [2] recently demonstrated that in many scenarios, random testing is more predictable and cost-effective at reaching high levels of structural coverage than previously thought.

Most studies concerning automated test generation for structural coverage maximization are focused on how to generate tests quickly or on improving coverage [1]. Comparisons of the fault finding effectiveness of the resulting test suites against other methods of test generation are few. Those that exist apart are largely, to the best of our knowledge, studies in concolic execution [12]—a combination of random testing with symbolic execution.

Recently, we have conducted a large-scale case study to evaluate the effectiveness of test suites generated to to satisfy branch and MC/DC coverage using counterexample-based test generation and a random generation approach, contrasted against purely random test suites of equal size [13]. Our results yielded two key conclusions. First, coverage criteria satisfaction alone is a poor indication of fault finding effectiveness, with random test suites of equal size providing similar—and often higher—levels of fault finding. Second, the use of structural coverage as a supplement—rather than a target—for test generation can have a positive impact, with random test suites reduced to a coverage-providing subset detecting up to 135% more faults than test suites generated to achieve coverage. A typical result from this study is illustrated in Figure 1, where for the three coverage criteria, random tests and a random test suite reduced to a coverage-satisfying subset outperform test suites automatically generated to achieve coverage of the same size.

3. KEY ISSUES

The existing body of research on this topic leads us to the observations that *it is not sufficient to simply maximize a structural coverage metric when automatically generating test inputs*, and therefore in practice *how coverage is achieved matters.*

The key underlying issues relate to how structural coverage criteria are formulated and how automatic test generation tools operate. Traditional coverage criteria are formulated over specific elements in the source code. To cover an element, (1) execution must reach the element and (2) exercise the element in a specific way. However, commonly used structural coverage criteria typically leave a great deal of leeway to how the element is reached, and—more importantly, in our experience—place no constraints whatsoever on how the test should evolve after the element is exercised. Automatic test generation tools typically use this freedom to do just enough work to satisfy coverage criteria, without consideration of, for example, how the faults are to be detected by the test oracle.

First, let us consider the path to satisfy a coverage obligation, e.g., a branch of a complex conditional. Structural coverage criteria require only that the point of interest is reached and exercised. We have found that automatically generated tests often take a shortest-path approach to satisfying test obligations, and manipulate only a handful of input values, leaving other inputs at default values. This is a cost effective method of satisfying coverage obligations; why tinker with program values that do not impact the coverage achieved? However, we and other authors have observed that variations provided by (for example) simple random testing result in test suites which are nearly as effective in terms of coverage, and moreover produce more interesting behavior capable of detecting faults [2, 13]. As illustrated in Figure 1, even with less coverage achieved, randomly generated test inputs can outperform automatically generated test suites in terms of fault finding.

Second, for the most commonly used structural coverage criteria, there is no directive concerning how tests should evolve after satisfying the structural element. Given this lack of direction, automatic test generation tools typically do not consider the path from the covered element to an output/assertion/observable state when generating a test, and therefore test inputs may achieve high coverage but fail to demonstrate faults that exist within the code. One reason for this failure is *masking*, which occurs when expressions/computations in the system are prevented from influencing the observed output, i.e., do not reach a variable or assertion monitored by the test oracle. More generally, this is related to the distinction between incorrect program state and program output: just because a test triggers a fault, there is no guarantee this will manifest as a *detected* fault. Indeed, in our experience, care must be taken to ensure that this occurs.

These two high level issues result in the generation of test inputs that may indeed be effective at encountering faults, but may make actually observing them—that is, actually detecting the fault—very difficult or unlikely. This reduces the effectiveness of any testing process based on structural coverage criteria, as we can easily satisfy coverage obligations for internal expressions without allowing resulting errors to propagate to the output.

We believe that these issues raise serious concerns about the efficacy of coverage-directed automated testing. As mentioned, the focus in automatic test generation work is currently on efficiently achieving coverage without carefully considering how achieving coverage impacts fault detection. We therefore run the risk of producing tools that are satisfying the letter of what is expected in testing, but not the spirit. Nevertheless, the central role of coverage criteria in testing is unlikely to fade, as demonstrated by the increasing use of coverage criteria for certification.

Hence, the key is to improve upon the base offered by these criteria and existing technology. We have come to the conclusion that the research goal in search-based test generation should not be developing methods of maximizing structural code coverage, but rather determining how to *maximize fault-finding effectiveness*. We propose that test generation is, in fact, a multi-objective optimization problem, and that algorithms built for test generation must evolve to take into account factors beyond the naive execution of individual elements of the code—factors such as masking, program structure, and the execution points monitored by the test oracle.

4. RECOMMENDATIONS

To support the broader objective of effective testing, we believe that search-based test generation approaches must evolve to take into account multiple factors—including not only satisfaction of structural coverage criteria, but also the propagation of faults to execution points monitored by the test oracle, and the underlying structure of the system under test. Given the two core issues identified in Section 3, we recommend two approaches.

First, we could improve, or replace, existing structural coverage criteria, extending them to account for factors that influence test quality. Automated test generation has improved greatly in the last decade, but the targets of such tools have not been updated to take advantage in this increase in power. Instead, we continue to rely on criteria that were originally formulated when manual test generation was the only practical method of ensuring 100% coverage.

Second, search-based test generation tools could be improved to avoid pitfalls when using structural coverage criteria. This could take many forms, but one straightforward approach would be to develop additional heuristics or rules that could operate alongside existing structural coverage criteria. For instance, tools could be encouraged to generate longer test cases, increasing the chances that a corrupted internal state would propagate to an observable output (or other monitored variable). Also, important factors specific to individual domains, e.g. web testing vs embedded systems, could be empirically identified and formalized as heuristics.

4.1 Use More Robust Coverage Criteria

In our own work, the issues of criteria formulation and masking motivated the development of the Observable MC/DC coverage criterion [16]. OMC/DC coverage explicitly avoids issues related to masking by requiring that its test obligations both demonstrate the independent impact of a condition on the outcome of the decision statement, and follow a non-masking propagation path to some variable monitored by the test oracle. OMC/DC, by accounting for the program structure and the selection of the test oracle, can, in our experience, address some of the failings of traditional structural coverage criteria within the avionics domain, allowing for the generation of test suites achieving up to 26% better fault detection than random test suites of equal size. Similarly, Vivanti et.al have demonstrated evidence that the use of data-flow coverage as a goal for test generation results in test suites with higher fault detection capabilities than suites generated to achieve branch coverage [15].

The primary problem with many existing structural coverage criteria is that all effort is expended on covering structures internal to the system, and no further consideration is paid to how the effect of covered structure reaches an observable point in the system. These results indicate that it is possible to overcome some of the issues we have highlighted by working with stronger coverage criteria or extending existing criterion to take into account issues such as fault propagation. OMC/DC considers the observability aspect for Boolean expressions (using MC/DC) by appending a path condition onto each test obligation in MC/DC. Similar extensions could be applied to a variety of other existing coverage metrics, e.g., boundary value testing.

4.2 Algorithmically Improve Test Selection

Extensions to coverage criteria are not without downsides: for stronger metrics, programs will contain *unsatisfiable* obligations where there is no test that can be constructed to satisfy the obligation. Depending on the search strategy, the test generator may never terminate on such obligations. Further, the higher cost and difficulty of generating OMC/DC satisfying test suites—relative to generating the weaker branch, condition, or MC/DC test suites—makes the use of strong coverage criteria as targets for test generation harder to universally recommend.

Instead, another possible method of ensuring test quality is to use a traditional structural coverage metric as the objective of test generation, and augment this by considering other factors empirically established to impact fault detection effectiveness. In the context of search-based test generation, this means adding additional objective functions to the search strategy, rather than adding additional constraints. For instance, an algorithm could both work to maximize an existing structural coverage criterion and minimize the propagation distance between the assignment of a value and its observation by the oracle (an algorithm that minimizes this distance for test prioritization purposes already exists [14]).

As an example, consider purely random testing. Random testing does not employ an objective function, but it is possible to use one to approximate how well the system's state space is being covered.We have seen systems containing bottlenecks (pinch-points) in the state space where randomly generated tests perform very poorly. Such bottlenecks require certain specific input sequences to reach a large portion of the state space. However, approaches such as concolic testing [12]—combining random testing with symbolic execution—are able to direct the generation of tests around such bottlenecks. Similar approaches could be employed to direct test generation towards, for example, propagation paths from variable assignment to oracle-monitored portions of the system.

While work exists considering test suite generation as a multi-objective problem, we believe that there is still much need for research in this area. By pursuing multiple objectives, we could potentially offer stronger tests that satisfy MC/DC obligations, even when it would be impossible to generate a test that satisfies the corresponding—but stricter—OMC/DC obligation. Embedding aspects of test selection into the objective function—or even into the search algorithm itself—may allow for improved efficiency. The search method can use, say, masking as a pruning mechanism on paths through the system, and the algorithm would not have to track as much symbolic information related to the objective metric itself.

4.3 Tailor an Approach to the Domain

It is important to emphasize that there is no "one size fits all" solution to test generation. The size and shape of the state space of a system varies dramatically between domains and programming paradigms, and, as a result, it is difficult to tailor universal testing strategies. Much of our work has focused on embedded systems that run as cyclic processes. In this area, a common issue is that the impact of exercising a code path on the system's output is delayed; only several cycles after a fault occurs can we observe it. If the goal of test generation is only to cause the code path to be executed, many of the tests will not cause a visible change in system behavior. In object-oriented systems, a central, but related issue is that a method call may change internal state that, again, is not visible externally until another method call produces output. As a result, choosing appropriate method sequences and their ordering becomes a major challenge.

Thus while general rules and heuristics for improving test generation are valuable, we believe there are large improvements to be found in tailoring the approach to the testing challenge at hand. For example, two often overlooked factors are the cost of generating tests and the cost of running tests. It is hard to outperform random testing in terms of the cost of generating tests, because doing so requires very little computation. If it is also cheap to run tests, then for many systems it is difficult to outperform straight random testing. On the other hand, if it is expensive to run tests, e.g., for embedded systems, this may require access to a shared hardware 'rig'. In this case, using search-based techniques to generate tests for a specific strong coverage criterion (such as OMC/DC) may be very sensible, because the number of required tests can be dramatically smaller than the number of random tests required to achieve the same level of fault finding.

Another overlooked factor is the "reasonableness" of generated tests. Automated test generation methods should deliver tests that not only find faults, but are also meaningful to the domain of interest. Coverage-based techniques take the path of least resistance when generating tests, but can produce tests that make little sense in the context of the domain. Techniques that can generate inputs with meaning to the human testers are valuable in reducing the "human oracle cost" associated with checking failing test results [10].

Addressing factors such as these must be done on a per domain level, and indicate that code coverage should be one of several goals in test generation. We suspect that, in the long run, effective automatic test generation tools will consist of both general techniques and heuristics and additional *test generation profiles* for each domain. Determining the *correct* objective functions for test generation for each domain is still an open research question, one requiring both technical advancements in search-based test generation and empirical studies.

5. CONCLUSION

While coverage criteria are a central aspect of software testing, both in practice and in research, there is an increasing body of work indicating that high structural coverage alone does not lead to effective testing—how the tests are generated also matters. To achieve effective testing, we should consider other factors, both general (e.g. test case length, path to test oracles) and domain specific.

Search-based test generation approaches, with their use of potentially multiple objective functions to guide search, are particularly well-suited to moving beyond simple structural coverage. We therefore believe that careful selection of the algorithms employed, the tuning of the parameters, and the objective functions used all have important roles to play. Moving in this direction requires work both empirical, determining exactly what factors influence fault detection the most for each software domain, and technical, translating that empirically acquired knowledge into heuristics and objective functions.

6. REFERENCES

[1] S. Anand, E. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, and P. McMinn. An orchestrated survey on automated software test case generation. *Journal of Systems and Software*, 86(8):1978–2001, August 2013.

[2] A. Arcuri, M. Z. Z. Iqbal, and L. C. Briand. Formal analysis of the effectiveness and predictability of random testing. In *ISSTA*, pages 219–230, 2010.

[3] T. Chen and Y. Yu. On the expected number of failures detected by subdomain testing and random testing. *IEEE Transactions on Software Engineering*, 22(2), 1996.

[4] A. Dupuy and N. Leveson. An empirical evaluation of the MC/DC coverage criterion on the hete-2 satellite software. In *Proc. of the Digital Aviation Systems Conference (DASC)*, Philadelphia, USA, October 2000.

[5] P. Frankl and S. N. Weiss. An experimental comparison of the effectiveness of the all-uses and all-edges adequacy criteria. In *Proc. of the Symposium on Testing, Analysis, and Verification*, 1991.

[6] W. Gutjahr. Partition testing vs. random testing: The influence of uncertainty. *IEEE Transactions on Software Engineering*, 25(5):661–674, 1999.

[7] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments of the effectiveness of dataflow-and controlflow-based test adequacy criteria. In *Proc. of the 16th Int'l Conf. on Software Engineering*. IEEE Computer Society Press Los Alamitos, CA, USA, 1994.

[8] N. Juristo, A. Moreno, and S. Vegas. Reviewing 25 years of testing technique experiments. *Empirical Software Engineering*, 9(1):7–44, 2004.

[9] S. Kandl and R. Kirner. Error detection rate of MC/DC for a case study from the automotive domain. *Software Technologies for Embedded and Ubiquitous Systems*, pages 131–142, 2011.

[10] P. McMinn, M. Stevenson, and M. Harman. Reducing qualitative human oracle costs associated with automatically generated test data. In *Proceedings of the First International Workshop on Software Test Output Validation*, STOV '10, pages 1–4, New York, NY, USA, 2010. ACM.

[11] A. Namin and J. Andrews. The influence of size and coverage on test suite effectiveness. In *Proc. of the 18th Int'l Symp. on Software Testing and Analysis*. ACM, 2009.

[12] K. Sen, D. Marinov, and G. Agha. CUTE: A concolic unit testing engine for C. In *Proc. of the 10th European Software Engineering Conf. / 13th ACM SIGSOFT Int'l. Symp. on Foundations of Software Engineering*. ACM New York, NY, USA, 2005.

[13] M. Staats, G. Gay, M. W. Whalen, and M. P. Heimdahl. On the danger of coverage directed test case generation. In *15th International Conference on Fundamental Approaches to Software Engineering (FASE)*, April 2012.

[14] M. Staats, P. Loyola, and G. Rothermel. Oracle-centric test case prioritization. In *ISSRE*, pages 311–320, 2012.

[15] M. Vivanti, A. Mis, A. Gorla, and G. Fraser. Search-based data-flow test generation. In *ISSRE'13: Proceedings of the 24th IEEE International Symposium on Software Reliability Engineering*. IEEE Press, Nov. 2013.

[16] M. Whalen, G. Gay, D. You, M. Heimdahl, and M. Staats. Observable modified condition/decision coverage. In *Proceedings of the 2013 International Conference on Software Engineering*. ACM, May 2013.