Multifaceted Test Suite Generation Using Primary and Supporting Fitness Functions *

Gregory Gay
University of South Carolina
Columbia, SC, United States
greg@greggay.com

ABSTRACT

Dozens of criteria have been proposed to judge testing adequacy. Such criteria are important, as they guide automated generation efforts. Yet, the current use of such criteria in automated generation contrasts how such criteria are used by humans. For a human, coverage is part of a *multifaceted* combination of testing strategies. In automated generation, coverage is typically *the* goal, and a single fitness function is applied at one time. We propose that the key to improving the fault detection efficacy of search-based test generation approaches lies in a targeted, multifaceted approach pairing *primary* fitness functions that effectively explore the structure of the class under test with lightweight *supporting* fitness functions that target particular scenarios likely to trigger an observable failure.

This report summarizes our findings to date, details the hypothesis of primary and supporting fitness functions, and identifies outstanding research challenges related to multifaceted test suite generation. We hope to inspire new advances in search-based test generation that could benefit our software-powered society.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; **Search-based software engineering**; *Software verification and validation*;

KEYWORDS

Automated Test Generation, Search-Based Test Generation, Adequacy Criteria

ACM Reference Format:
Gregory Gay. 2018. Multifaceted Test Suite Generation Using Primary and Supporting Fitness Functions . In *SBST'18: SBST'18:IEEE/ACM 11th International Workshop on Search-Based Software Testing , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3194718.3194723

*This work is supported by National Science Foundation grant CCF-1657299.

SBST'18, May 28–29, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5741-8/18/05…$15.00
https://doi.org/10.1145/3194718.3194723

1 INTRODUCTION

With exponential growth in the cost of software testing, we must find means of reducing costs while maintaining quality. Automation of tasks such as unit test creation has a critical role to play in reducing testing costs [2]. Yet, despite advances in automated test generation technology, the efficacy of the produced test suites at detecting faults has yet to match human-produced tests [6, 8, 10].

As we cannot know what faults exist a priori, dozens of criteria—ranging from the measurement of structural coverage to the detection of synthetic faults [9]—have been proposed to judge testing *adequacy*. In theory, if the goals set forth by such criteria are fulfilled, tests should be *adequate* at detecting faults related to the focus of that criterion. Adequacy criteria are important for search-based generation, as they are the most common basis for the fitness functions that judge solutions and guide the search.

Yet, the current use of adequacy criteria in automated generation sharply contrasts how such criteria are used by humans. For a human, coverage typically serves an advisory role—as a way to point out gaps in existing efforts. Human testers build suites in which adequacy criteria contribute to a *multifaceted* combination of testing strategies. Yet, in automated generation, coverage is typically *the* goal, and a single fitness function is applied at one time. Yet, search-based techniques need not be restricted to one criterion—one fitness function—at a time. The test obligations of multiple criteria can be combined into a single score or simultaneously satisfied by multi-objective optimization algorithms. Such *multifaceted* suites have the potential to be more effective than those generated using a single fitness function, as they trade a laser-focus on that one criterion for reasonably high coverage of a varied set of goals [9].

In previous work, we have explored the efficacy of both individual fitness functions [6] and combinations of functions [7] at detecting real faults, given a fixed search budget. Our observations have revealed that, while certain fitness functions are more effective than others, almost all functions are *situationally* adept at detecting certain types of faults. Further, we have found that both high coverage of class structure *and* high satisfaction of the goals of the chosen fitness function are both needed to detect faults. Combining situationally-adept functions like Exception or Output Coverage—particularly functions that lack their own means to increase structural coverage—with a strong structure-focused function such as Branch Coverage—which is unable on its own to favor targeted fault types—yields significant improvements in the likelihood of fault detection. Well-chosen fitness functions, when used in combination, are able to guide the test generation framework towards test suites that combine the strengths of each function, overcome their weaknesses, and produce a testing strategy that is more effective than any single function—even without an increase in search budget.

Therefore, we propose that the key to improving the fault detection efficacy of search-based test generation approaches lies in a *human-like* approach to test creation—the application of a targeted, multifaceted approach to generation where multiple testing strategies are selected and simultaneously explored. We hypothesize that effective test generation strategies will pair **primary** fitness functions that effectively exploit the structure of the class under test with lightweight **supporting** fitness functions that target particular aspects of the class under test likely to trigger an observable failure.

We propose that the hypothesis of effective multifaceted generation based on primary and supporting fitness functions should be explored by the search-based software testing community, and that numerous research challenges connected to this hypothesis remain to be solved. Advances are needed in terms of how criteria are optimized, the identification of new supporting fitness functions tied towards particular types of software faults, selection of a multifaceted function portfolio for new classes and systems, and approaches that reduce the difficulty of generation under a limited budget.

This report summarizes our findings to date, details the hypothesis of primary and supporting fitness functions, and identifies outstanding research challenges related to the topic of multifaceted test suite generation. We hope to inspire new advances in search-based test generation that have the potential to impact industrial practices and benefit our software-powered society.

2 PRELIMINARY RESULTS

We have previously performed empirical studies on the ability of individual criteria to produce test suites that detect real faults [6][1]. After assessing such suites on 593 faults from 15 open-source Java projects, we have found that:

- Branch Coverage[2] detects more faults and demonstrates a higher likelihood of detection than other functions, given a fixed search budget.
- Regardless of overall performance—most functions have situational applicability, where suites detect faults no other function can detect. Exception[3], Output[4], and Weak Mutation Coverage[5] show situational applicability, even if their average efficacy is lower than Branch Coverage.
- Factors that indicate a high level of efficacy include high structural coverage over the code and high coverage of the chosen function's test obligations. In situations where achieved structural coverage is low, the fault does not tend to be found.
- The factor that differentiates occasionally detection and *consistent* detection of a fault is satisfaction of the chosen function's test obligations. The best suites are ones that *both* explore the code and fulfill their own testing goals, which may be—in cases such as Exception Coverage—orthogonal to structural coverage.

We have also performed exploratory studies of the fault-detection capabilities of combinations of criteria [7]. We have observed:

- A combination of all eight studied functions performs well, but the difficulty of simultaneously satisfying all functions prevents it from outperforming every individual function under a fixed search budget. However, for all systems, at least one targeted combination is more effective than every individual function.
- The most effective combinations vary by system, but all pair a structure-focused function—such as Branch Coverage—with supplemental strategies targeted at the class under test.
  - Across the board, effective combinations include Exception Coverage. Method Coverage[6] also generally offers an efficacy boost. Both can be added to a combination with minimal effect on generation complexity.
  - Additional targeted criteria—such as Output Coverage for code that manipulates numeric values or Weak Mutation Coverage for code with complex logical expressions—offer further efficacy improvements.

3 PRIMARY AND SUPPORTING FITNESS FUNCTIONS

Fitness functions represent strategies that can be used to manipulate the search. In single-objective generation, the chosen fitness function will determine the focus, strengths, and weaknesses of the resulting suite. If multiple fitness functions are simultaneously applied—whether combined into a single score or simultaneously optimized by a multi-objective algorithm—the resulting test suite will be the product of the interaction of the chosen strategies. In theory, the simultaneous use of a *portfolio* of fitness functions during generation could produce test suites that are able to detect a variety of faults—trading precise focus on one fitness function for reasonable coverage of multiple functions [9].

In practice, selecting this portfolo of fitness functions requires careful consideration. Given a limited fixed time limit for generation, the framework will more easily achieve high coverage of a single function than high coverage of multiple functions, as the combination will require that more goals be met—and, at times, that conflicting goals be met. This explains why the eight-way combination of functions used by the EvoSuite framework is often outperformed by individual functions [6, 7]. The difficulty of optimizing for so many functions in the same time window alloted to one function led to weaker results. Given a sufficiently-long generation period, that eight-function combination may produce stronger test suites. However, our observations also indicate that careful selection of a fitness function portfolio can yield superior results *without increasing the search budget*. We hypothesize that effective automated generation may be performed through targeted selection of this portfolio.

In general, faults cannot be detected without executing the affected lines of code. This is why structure-based criteria such as Branch Coverage and Line Coverage dominated our ranking of individual fitness functions. Yet, past research also indicates that *coverage alone is not enough* [8]. Merely executing a line of code in *any* manner is not sufficient to detect a fault. *How* that line of code is executed matters. In our experiments, a consistently high likelihood of fault detection requires both coverage of the code structure *and* coverage of a fitness function's goals. In the case of functions

[1] Due to space constraints, we do not fully define each fitness function here. Full definitions can be found in: [6]
[2] Branch Coverage requires that each control-altering decision outcome be exercised.
[3] Exception Coverage rewards suites that cause more exceptions to be thrown.
[4] Output Coverage rewards coverage of type-specific abstract output values
[5] Weak Mutation Coverage rewards coverage of synthetic faults.
[6] Method Coverage requires that each method be called by test cases.

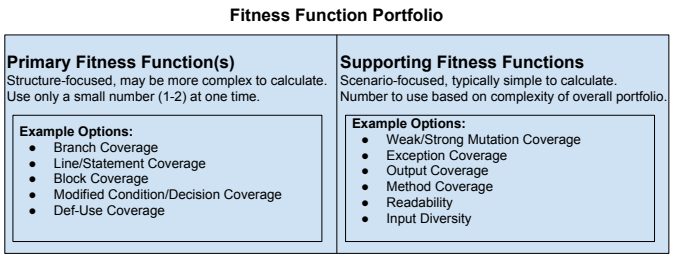

Figure 1: Example primary and secondary fitness functions.

such as Branch or Line Coverage, these two are the same. However, targeted fitness functions such as Exception or Output Coverage lack an in-built means to drive structural coverage. This may limit the efficacy of such functions as the sole target of test generation, but illustrates a potential advantage of multifaceted generation. A structure-focused function could be used to explore execution paths, while targeted fitness functions could be simultaneously used to *shape* that exploration process—tuning the resulting test suite.

Exception Coverage is effective because it rewards suites that trigger more exceptions—which often are the observable manifestation of a fault. However, it lacks any feedback mechanism to drive generation towards exceptions. Branch Coverage is effective at exploring the structure of a class, but lacks the context needed to drive execution to an observable failure. By uniting the two, Branch Coverage provides the means to explore the class under test—leading to the discovery of additional exceptions. Exception Coverage provides a weighting mechanism to Branch Coverage—increasing the odds of detecting a fault. We hypothesize then, that the key to effective testing is the identification of the correct portfolio of **primary and supporting fitness functions** for the class or system under test.

Primary fitness functions are designed to explore the structure of the class under test. Common examples of such criteria include Branch, Line, or MC/DC Coverage. As structural coverage is a prerequisite to fault detection, primary functions should be the focus of the generation portfolio—perhaps even weighted more heavily than other fitness functions. As such criteria tend to be more complex to compute, requiring more execution time to calculate than other criteria, a limited number of primary functions should appear in the portfolio—typically only one. However, as we have observed scenarios where Branch and Line Coverage were more effective combined than in isolation, multiple primary functions may be considered.

Supporting fitness functions are intended to control *how* code is executed, and should be targeted towards scenarios or faults of interest. For example, Exception Coverage rewards suites that trigger more exceptions [9]. Output Coverage rewards suites that produce particular defined value types for output that belongs to particular data types [1]. Mutation Coverage rewards suites that detect synthetic faults [5]. Such fitness functions—on their own—do not necessarily have the means to drive structural coverage. However, when paired with primary criteria, they can manipulate the overall search strategy, increasing the likelihood of detecting certain types of faults. As such fitness functions must rely on complex primary functions for code exploration, supporting criteria should be *lightweight*. They should not unduly increase the time required to calculate fitness. The number to be used should be based on the complexity of the overall portfolio, and should remain relatively low.

We hypothesize that a well-chosen portfolio of primary and supporting functions will be able to shape test generation in a more human-like manner—resulting in test suites tuned towards particular types of systems, faults, or testing scenarios. However, a number of research challenges related to the selection and optimization of primary and supporting fitness functions remain unsolved.

4 RESEARCH CHALLENGES

4.1 How to Optimize The Chosen Functions

The portfolio of fitness functions can be optimized in multiple ways. Some techniques, such as EvoSuite, combine multiple fitness functions into a single score [9]. Individual fitness functions can be weighted, but an improvement in *any* of the chosen functions is considered to be an improvement in the overall score. A loss of score for one function is acceptable if balanced by enough improvement in another. Other techniques, such as jMetal [4], attempt to simultaneously optimize separate fitness functions. Such approaches attempt to find an optimal balance between each distinct fitness functions.

A portfolio of primary and supporting functions can be explored using either approach. However, each will produce distinct test suites. Merging each function into a single overall score could result in a suite that heavily favors a particular function. However, this may actually be desirable—for example, we may prefer a suite that attains higher structural coverage and lower coverage of a supporting function over a suite that attains perfect balance of the primary and supporting criteria. Different means of optimizing the portfolio should be explored to better understand how to generate suites.

4.2 Identification of the Criteria Portfolio

The identification of the correct portfolio of fitness functions for a system is not a trivial task. In our experience, the most effective portfolio varies from system to system, and depends on the purpose of the system and the type of mistakes that developers tend to make on that project [7].

Because they add little difficulty to generation, we have observed that Exception and Method Coverage have the most consistent effect on suite efficacy. However, targeted supporting criteria often had beneficial effects as well. For example, generation for the *Apache Commons Math* project typically benefited from the inclusion of Output Coverage. This project offers a variety of tools for numeric analysis, and Output Coverage's focus on different abstract types of numeric values naturally led to an increased rate of fault detection.

One way to choose the criteria portfolio for a new class could be through the use of reinforcement learning (RL) [11] as part of the generation process. Each round of generation, the RL algorithm could choose a new portfolio. After each choice, the algorithm would receive a reward chosen from a probability distribution dependent on the portfolio selected. Over time, it would attempt to maximize the total expected reward, identifying the portfolio most adept at improving a chosen "reward function." Then, the chosen combination could be used from the start of generation—without reinforcement learning—when testing that class in the future.

If developers seek to maximize coverage of a particular adequacy criterion—for instance, developers of avionics applications must satisfy MC/DC Coverage to attain safety certification [8]—then coverage of that criterion could serve as the reward function. The

RL algorithm would suggest a portfolio adept at quickly attaining MC/DC. If a particular criterion is chosen as the reward function, then the RL algorithm could suggest a portfolio that both meets a chosen testing goal and is still able to detect a variety of faults. This process, in particular, could be quite useful for maximizing criteria that are too complex to serve as ideal fitness functions. For example, Strong Mutation Coverage—which requires input that reveals the presence of synthetic faults at the output level—is difficult to optimize as a direct fitness function as it offers little feedback to the search. However, the RL process could suggest a portfolio adept at achieving high levels of Strong Mutation Coverage—made up of individual fitness functions that do offer feedback to the search.

4.3 Discovery of New Supporting Functions

If the structural coverage enabled by the use of a primary fitness function enhances the efficacy of supporting functions, then new fitness functions can be formulated and experimented with without the need for an in-built coverage mechanism. This opens the opportunity to craft new fitness functions around particular testing scenarios, fault types, or other measures that could enhance the fault-detection capabilities of the generated test suites.

For example, when generating test suites, a set of classes must be chosen as targets. Often, generation is performed solely on classes whose code was directly changed. However, the likelihood of fault detection could be increased by also targeting classes that are *coupled*—dependent—on those changed classes. This is particularly true during regression or integration testing. A supporting fitness function could reward test suites that cover connections between the current generation target and particular classes of interest. By relying on structural coverage from a primary criterion, this function could be efficiently calculated as a count of dependencies covered. This would do little to increase the difficulty or cost of generation.

Similarly, many approaches to test generation reward input diversity [3]. Such approaches ensure a wide spread of input choices over the possible option space, theorizing that ensuring diversity will improve the likelihood of fault detection. As a supporting fitness function, a simple diversity measurement could weight the selection of input chosen by the primary fitness function. This would help ensure that a variety of input choices are used while still ensuring a high level of code coverage. This simple metric could be added to a portfolio of other supporting functions as a low-cost means to further increase the likelihood of detection.

4.4 Improving Generation Efficiency

Fundamentally, it is more difficult to generate a suite that optimizes multiple fitness functions than a suite that optimizes a single function. At the least, the task presented to the generation framework is more difficult. Improvements in the efficiency of generation are needed. Such improvements will benefit generation for both single functions and multifaceted portfolios. In that case, more time could be allocated to multifaceted generation without unduly delaying the testing process. In addition, improvements in the efficiency of calculating fitness for individual criteria will also benefit the ability to generate for multiple criteria.

Another potential avenue for improvement could be similar in form to the *archiving* of test obligations performed by EvoSuite. When obligations are fully satisfied for criteria such as Branch Coverage in EvoSuite, they can be removed from the overall fitness evaluation. This enables improvements in efficiency, as fitness becomes progressively faster to calculate. A similar process could be used to stagger the inclusion of additional criteria. A small "core" portfolio could be considered at the beginning of the generation process. As obligations are covered, additional criteria could be incorporated. Over time, this process would reshape the population of test suites to steadily cover additional facets, and could reduce the complexity of fitness calculation at any one step in that process.

5 CONCLUSIONS

We propose that the key to improving the fault detection efficacy of search-based test generation approaches lies in a *human-like* approach to test creation—the application of a targeted, multifaceted approach to generation where multiple testing strategies are selected and simultaneously explored. We hypothesize that effective test generation strategies will pair strong *primary* fitness functions—criteria that effectively exploit the structure of the class under test—with lightweight *supporting* fitness functions that target particular aspects of the system under test likely to trigger an observable failure.

We hope to inspire new advances in mutifaceted test generation that could impact industrial practices and benefit our software-powered society. In particular, advances are needed in how we select a portfolio of fitness functions, new fitness functions are needed that target a variety of testing scenarios, and improvements must be discovered in regards to generation difficulty and time requirements.

REFERENCES

[1] N. Alshahwan and M. Harman. Coverage and fault detection of the output-uniqueness test selection criteria. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, pages 181–192, New York, NY, USA, 2014. ACM.

[2] 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.

[3] T. Chen, H. Leung, and I. Mak. Adaptive random testing. In M. Maher, editor, *Advances in Computer Science - ASIAN 2004. Higher-Level Decision Making*, volume 3321 of *Lecture Notes in Computer Science*, pages 3156–3157. Springer Berlin / Heidelberg, 2005.

[4] J. J. Durillo and A. J. Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760 – 771, 2011.

[5] G. Fraser and A. Arcuri. Achieving scalable mutation-based generation of whole test suites. *Empirical Software Engineering*, 20(3):783–812, 2014.

[6] G. Gay. The fitness function for the job: Search-based generation of test suites that detect real faults. In *Proceedings of the International Conference on Software Testing*, ICST 2017. IEEE, 2017.

[7] G. Gay. Generating effective test suites by combining coverage criteria. In *Proceedings of the Symposium on Search-Based Software Engineering*, SSBSE 2017. Springer Verlag, 2017.

[8] G. Gay, M. Staats, M. Whalen, and M. Heimdahl. The risks of coverage-directed test case generation. *Software Engineering, IEEE Transactions on*, PP(99), 2015.

[9] J. M. Rojas, J. Campos, M. Vivanti, G. Fraser, and A. Arcuri. Combining multiple coverage criteria in search-based unit test generation. In M. Barros and Y. Labiche, editors, *Search-Based Software Engineering*, volume 9275 of *Lecture Notes in Computer Science*, pages 93–108. Springer International Publishing, 2015.

[10] S. Shamshiri, R. Just, J. M. Rojas, G. Fraser, P. McMinn, and A. Arcuri. Do automatically generated unit tests find real faults? an empirical study of effectiveness and challenges. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE 2015, New York, NY, USA, 2015. ACM.

[11] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.