Using Synthetic Test Suites to Empirically Compare Search-Based and Greedy Prioritizers

Zachary D. Williams
Department of Computer Science
Allegheny College
williaz@allegheny.edu

Gregory M. Kapfhammer
Department of Computer Science
Allegheny College
gkapfham@allegheny.edu

ABSTRACT

The increase in the complexity of modern software has led to the commensurate growth in the size and execution time of the test suites for these programs. In order to address this alarming trend, developers use test suite prioritization to reorder the test cases so that faults can be detected at an early stage of testing. Yet, the implementation and evaluation of greedy and search-based test prioritizers requires access to case study applications and their associated test suites, which are often difficult to find, configure, and use in an empirical study. This paper presents two types of synthetically generated test suites that support this process of experimentally evaluating prioritization methods. Using synthetic test suites affords greater control over test case characteristics and supports the identification of empirical trends that contradict the established wisdom about search-based and greedy prioritization. For instance, we find that the hill climbing algorithm often exhibits a lower time overhead than the greedy test suite prioritizer while producing test orderings with comparable effectiveness scores.

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

General Terms: Experimentation, Performance

Keywords: search-based and greedy test prioritization

1. INTRODUCTION

Software developers often introduce defects during the implementation process. Regression testing methods establish a confidence in the correctness of and isolate defects within a program by running a collection of tests known as a test suite. Since regression testing can be very time consuming, testers use search-based and greedy prioritization techniques to produce a test ordering that will reveal faults earlier in the suite's execution than would otherwise be possible.

Suppose that a test suite $T = \langle t_1, t_2, t_3, \ldots, t_n \rangle$ covers the set of requirements $\mathcal{R}(T) = \{r_1, r_2, r_3, \ldots, r_m\}$. Each test case $t_i \in T$ is associated with the non-empty set $\mathcal{R}(t_i) \subseteq \mathcal{R}(T)$ [1, 5]. During the empirical study of search-based and greedy test suite prioritizers, researchers often use the T and $\mathcal{R}(T)$ associated with real-world case study applications. Yet, this practice can be difficult and time consuming because of the need to tailor prototype test reordering tools for complex real-world programs. Furthermore, some small case study applications may not be representative of all real-world programs, thus hindering empirical investigations of the efficiency and effectiveness of approaches to testing. The use of real-world programs also prohibits the experimenter from easily controlling the size of the test suite T and the coverage patterns within $\mathcal{R}(T)$. Ultimately, the lack of a wide variety of test suites hinders the ability of researchers to quickly compare and contrast the plethora of newly developed techniques for test prioritization (e.g., [1, 3, 4, 5]).

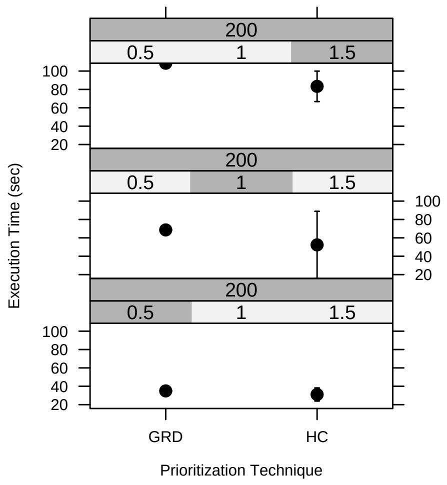

Figure 1: Execution Time - Fully Random.

Synthetic Test Suites. Using efficiently generated synthetic test suites to study search-based and greedy prioritizers enables experimenters to easily establish baseline results and control the key characteristics of the tests [2]. As such, this paper describes two simple methods for generating synthetic test suites and demonstrates how they reveal fundamental trade-offs in test prioritization techniques. We used two parameters to create the synthetic test suites: requirement factor F_r and coverage point factor F_c. For a chosen test suite size, F_r controls how many requirements the generated test suite will have, such that $|\mathcal{R}(T)| = F_r \times |T|$. After setting the size of the test suite and requirement set, we use F_c to define the number of times the requirements are covered, denoted C, as a fraction of the total number of possible coverage points, so that $C = F_c \times |\mathcal{R}(T)| \times |T|$.

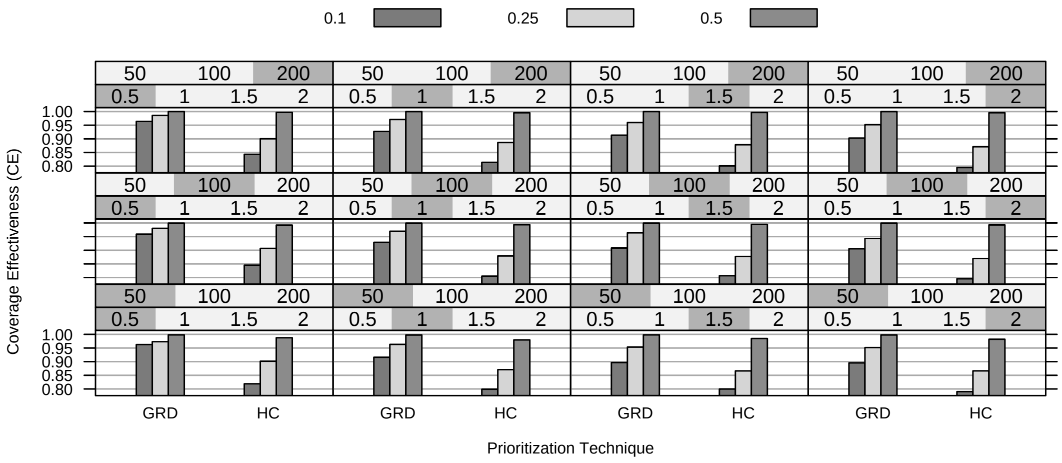

Figure 2: Coverage Effectiveness Values for all Coverage Point Factors - Controlled Random.

As an example, suppose that $T = \langle t_1, t_2, t_3, t_4 \rangle$ and we set $F_r = 1$, thereby stipulating that $\mathcal{R}(T) = \{r_1, r_2, r_3, r_4\}$. If we specify $F_c = 0.5$, then we know that $C = 0.5 \times 4 \times 4 = 8$ and thus $|\mathcal{R}(t_1)|+|\mathcal{R}(t_2)|+|\mathcal{R}(t_3)|+|\mathcal{R}(t_4)| = 8$. That is, we use F_c to designate that the four requirements will be covered a total of eight times, with each test covering roughly the same number of requirements. To complement this controlled approach that necessitates the use of two parameters, we also developed a fully random test suite generator that only uses F_r to manage the size of T and $\mathcal{R}(T)$ and allows the location and number of coverage points to vary randomly.

2. EMPIRICAL STUDY

Techniques. In order to demonstrate the utility of synthetically generated coverage reports we used them to study different prioritization methods. The experiment considered a search-based hill climber (HC) and a greedy algorithm (GRD) for prioritization because they are both simple to implement and relatively efficient to execute, thus making them widely used in practice [3]. These algorithms are a good choice for an initial comparison because prior experimental results showed that greedy methods commonly have little variability in time overhead and often produce effective test orderings, while searched-based methods may lack uniformity in their execution times and construct prioritizations with modest effectiveness ratings [1, 3, 5].

GRD iteratively selects the next-best test case according to a metric that computes the ratio of cost and coverage for each test, whereas HC employs a first ascent hill climbing approach [1, 3, 4]. HC uses either a “swap-first” or “swap-last” technique to generate a neighborhood for a given test suite. Swap-first switches the first test with each remaining test in the suite, thus creating a neighborhood of new orderings (with the exception of focusing on the last test, swap-last proceeds in an analogous fashion). After choosing a starting test suite, the first ascent (FA) algorithm finds the first neighbor with a higher effectiveness score and then generates a new neighborhood from that ordering. This process continues until no better ordering can be found. Picking swap-first, we consider FA instead of the steepest ascent (SA) method that examines every neighbor because FA is more efficient and only marginally less effective than SA [1].

Results. We used execution time to measure the efficiency of the prioritization techniques and selected the higher-is-better coverage effectiveness (CE) metric to gauge the quality of the prioritized suite [5]. For both the fully random and controlled random test suites, the empirical study reveals that HC was often faster than GRD for suites with more than 200 tests. Figure 1 shows this trend for the fully random suites, with the error bars in the graph denoting one standard deviation from the mean of the time to execute the prioritizers. The bars at the top of these graphs are “shingles” designed to provide information about the characteristics of the test suite. The upper level of shingles is the number of tests, $|T| = 200$, while the lower level shows the requirement factor, $F_r \in \{0.5, 1, 1.5\}$. This graph contradicts the popular notion that greedy algorithms are faster than search-based methods by showing that HC’s arithmetic means for prioritization time are lower than those of GRD.

As in Figure 1, the shingles in Figure 2 display $|T|$ on the top and F_r below, with each shaded bar representing a different coverage point factor $F_c \in \{0.1, 0.25, 0.5\}$. These graphs demonstrate that when $F_c = 0.1$ and thus each test only covers a few requirements, GRD is more effective than HC. We note that the same result generally holds for $F_c = 0.25$, although the trend is less pronounced. Yet, when F_c increases to 0.5, HC is competitive with GRD across a wide range of values for $|T|$ and $|\mathcal{R}(T)|$. Since Figure 1 shows that HC can often be more efficient than GRD, albeit with more variability, these results suggest that search-based methods may be gainfully employed by prioritizing moderate size suites that repeatedly cover the requirements. Thus, we intend to develop new generators and conduct additional experiments since these outcomes suggest that synthetic suites support the identification of meaningful trade-offs in the efficiency and effectiveness of search-based and greedy prioritizers.

3. REFERENCES

[1] A. Conrad, R. S. Roos, and G. M. Kapfhammer. Empirically studying the role of selection operators during search-based test suite prioritization. In *GECCO*, 2010.

[2] F. Haftmann, D. Kossmann, and E. Lo. A framework for efficient regression tests on database applications. *The VLDB Journ.*, 16(1), 2007.

[3] Z. Li, M. Harman, and R. M. Hierons. Search algorithms for regression test case prioritization. *Trans. on Softw. Eng.*, 33(4), 2007.

[4] G. Rothermel, R. J. Untch, and C. Chu. Prioritizing test cases for regression testing. *Trans. on Softw. Eng.*, 27(10), 2001.

[5] A. M. Smith and G. M. Kapfhammer. An empirical study of incorporating cost into test suite reduction and prioritization. In *SAC*, 2009.