# Interactive Coverage Effectiveness Multiplots For Analyzing Prioritized Regression Test Suites

Adam M. Smith* Joshua J. Geiger†

University of Pittsburgh

**ABSTRACT**

Software testing increases confidence in the correctness of an application's source code. Altering a test suite's execution order enables earlier detection of defects and allows developers to fix errors sooner. The many existing ordering methods produce different possible test suite orders to choose from. This paper presents tool support for a technique that allows for a comparison of test suite orders through visualization and interaction.

## 1 INTRODUCTION

Developers inevitably create errors while designing and implementing software systems. Software developers execute tests $\langle t_1, t_2, t_3, \ldots, t_n \rangle$ in a test suite $T$ to isolate defects and gain confidence in the correctness of the code. Each test case in the test suite exercises specific points in the system, comparing the actual output of the code to the hand computed expected one. If a test fails then it is likely that a defect is present in the source code that the test executes. As the source code grows in size and number of features new tests are written for the new functionality. To ensure that the new features do not cause the system to regress, developers include every previously written test in the collection of tests. This process of executing and re-executing the entire test suite is known as regression testing.

Gradually, adding new tests and retaining old ones increases the size of the test suite until its execution time can become prohibitively expensive. One method of altering the test suite to resolve this issue is test suite prioritization [4]. Prioritization attempts to find an ordering of the test cases that is more likely to locate defaults earlier in the execution of the test suite without risking the loss of coverage by removing a test. The tests are ordered based on certain criteria that are obtained during a process called coverage monitoring.

Coverage monitoring measures the code coverage of a test case $t_i$. Code coverage describes any metric that enumerates specific points in the source code that are executed when a test is run, whether they are a line, a block, a method, a branch [6], or another type. Each specific program point is called a requirement. Given a test suite $T$, coverage monitoring gives a set of requirements $\mathcal{R}(T) = \{r_1, r_2, \ldots, r_m\}$. Each individual test $t_i$ is associated with a subset of requirements $\mathcal{R}(t_i) \subseteq \mathcal{R}(T)$, which it is said to *cover*.

Table 1 shows an example of a test suite with 4 tests and 5 requirements. An X in a cell represents that the test for that row covers the requirement in that column. Consider running the shown test suite in its original order. In this case all of the requirements are not covered until 8 time units have passed. Conversely, if the test suite is executed in reverse order all of the requirements are covered in 4 time units. Covering all of the requirements sooner allows for a higher chance to find faults earlier so that the developer can more quickly begin to make changes.

| | $r_1$ | | $r_2$ | $r_3$ | $r_4$ | Execution Time |
|---|---|---|---|---|---|---|
| $t_1$ | X | X | X | X | | 4 |
| $t_2$ | | | X | X | | 1 |
| $t_3$ | | X | | | | 1 |
| $t_4$ | X | | | | X | 2 |

Table 1: Example Test Suite

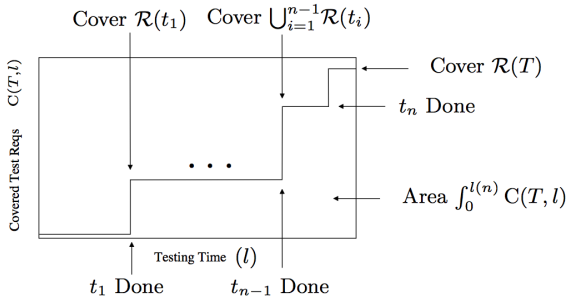


Figure 1: Coverage Effectiveness.

The metric coverage effectiveness (CE) [3] was developed in order to rate a test suite prioritization. CE is derived from the step function of the cumulative coverage of a test suite as shown in Figure 1. Each test suite offers the possibility of covering more total requirements. When the cumulative coverage is plotted against time a step function is formed. CE itself is calculated by dividing the area under the step function for the actual order by the area under the curve of a test suite that covers all of its requirements instantly. This value is inclusively between 0 and and less than 1 where a CE of 0 would mean that no requirements were covered and a CE of 1 would mean that all of the requirements were covered instantly.

## 2 MOTIVATION

For test suites with $n$ tests it is too expensive to generate all $n!$possible orderings to find the best CE value. For this reason there are several algorithms that prioritize test suites. Given coverage and timing data for a test suite this tool generates several prioritizations using the implementations of greedy (GRD), 2-optimal greedy (2OPT), delayed greedy (DGR), and Harrold Gupta Soffa (HGS) algorithms described in Smith and Kapfhammer [5]. These algorithms can use the execution time (cost), covered requirements (coverage), or a ratio of covered requirements per unit time (ratio) to make all necessary greedy choices.

In addition to algorithmic approaches to prioritization, random sampling may produce orderings that are favored over the original order. Generating large random samples also gives insight into the difficulty of finding a good ordering. The reverse order of a test suite also tends to produce higher CE values than the original.

It is difficult to interpret the results of several algorithms or random samples by reading text alone. Therefore, this paper presents

*e-mail: ams292@cs.pitt.edu

†e-mail:jj55@cs.pitt.edu

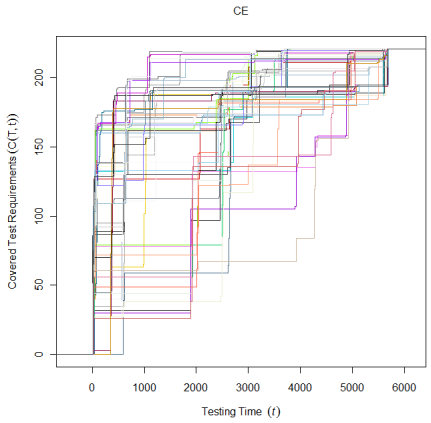


Figure 2: Coverage Effectiveness Multiplot

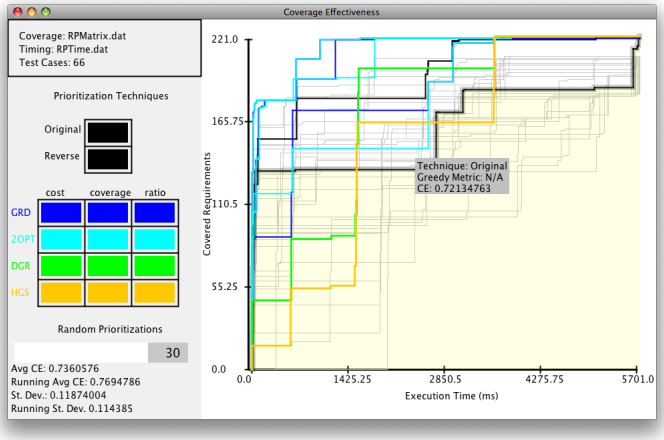


Figure 3: Interactive Multiplot

a visualization that can allow software testers to examine a large set of test suite orders simultaneously. Related work in the field of visualizing test suites exists [2], however, it focuses on fault localization or different features of a test suite.

Examining the CE functions of several orderings for a given test suite will reveal the effectiveness of the different prioritization techniques on that suite. Figure 2 shows a multiplot of CE functions for 50 random prioritizations. This static image does not allow for easy identification of the source of the prioritizations. Also, the static image cannot display qualitative data without using a large legend. To allow for more information communication the visualization provides interactivity in order to more easily allow test suite orders to be compared.

## 3 Tool Design

The tool is designed to aid in analyzing multiple prioritizations of a test suite through user interaction of CE multiplots. Due to the fact that step functions consist of only vertical and horizontal lines, plotting several functions on a single graph may obscure lines, thus making it difficult to track and compare prioritizations. A solution to this is pruning the graph to only the plots in which the user is interested. However, generating a new static multiplot for each comparison is cumbersome. Therefore, in a similar approach to Becker et al. [1] and a NY Times interactive visualization of market statstics [1] the tool allows users to interactively select techniques displayed and directly manipulate the corresponding plot. Figure 3 provides a screen capture of the tool.

The left panel provides information about the test suite and allows the user to select which technique's results will be displayed in the multiplot. The first set of buttons will display the original or the reverse order of the test suite. The button matrix toggles displaying the results of the prioritization algorithms introduced above. Each row represents a prioritization technique and each column shows a greedy choice metric. To display the results of a technique given a specific metric the user may click on the appropriate cell in the matrix. Each technique button is color coded to match its step function line in the plot for easy identification. The slider bar allows the user to choose a number of random prioritizations that are displayed in the plot as thinner gray lines.

The right panel displays the multiplot. In this image, all techniques are currently selected to appear in the multiplot. A mouseover on a function line highlights it and shades the area under it, as well as, provide a label identfying the technique, greedy choice metric, and the CE value of the prioritization represented by the line.

## 4 Evaluation

The tool support for this regression test suite visualization technique is available for download at *raise.googlecode.com*.

## References

[1] R. A. Becker, S. G. Eick, and A. R. Wilks. Visualizing Network Data. *IEEE Transactions on Visualization and Computer Graphics*, 1:16–28, 1995.

[2] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 467–477, New York, NY, USA, 2002. ACM.

[3] G. M. Kapfhammer and M. L. Soffa. Using coverage effectiveness to evaluate test suite prioritizations. In *Proceedings of the ACM Workshop on Empirical Assessment of Software Engineering Languages and Technologies*, November 2007.

[4] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Prioritizing Test Cases For Regression Testing. *IEEE Transactions on Software Engineering*, 27(10):929–948, 2001.

[5] A. M. Smith and G. M. Kapfhammer. An empirical study of incorporating cost into test suite reduction and prioritization. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 461–467, New York, NY, USA, 2009. ACM.

[6] H. Zhu, P. A. V. Hall, and J. H. R. May. Software unit test coverage and adequacy. *ACM Comput. Surv.*, 29(4):366–427, 1997.

[1] http://www.nytimes.com/interactive/2008/10/11/business/20081011_BEARMARKETS.html