

Interactive Coverage Effectiveness Multiplots For Evaluating Prioritized Regression Test Suites

Adam M. Smith*, Joshua J. Geiger, G. Elisabeta Marai
University of Pittsburgh

Manos Renieris
Google

Gregory M. Kapfhammer
Allegheny College

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 from which to choose. This paper presents an interactive visualization technique with tool support that enables easy comparisons of the coverage effectiveness of many test suite orders.

1 BACKGROUND

Developers inevitably create errors while designing and implementing software systems. Software developers often execute a test suite $T = \langle t_1, t_2, t_3, \dots, t_n \rangle$ in order to isolate defects and gain confidence in the correctness of a program. 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 as well. 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 [5].

Gradually, the addition of new tests increases the size of the test suite until its execution time may become prohibitively expensive. One method of altering the test suite to resolve this issue is test suite prioritization [5] [6]. Prioritization attempts to find an order of the test cases that is more likely to locate defaults earlier in the execution of the test suite. Test prioritizers use output from a coverage monitor which measures and enumerates the specific points in the source code that are executed when a test t_i is run. Tracking coverage points such as a line, block, method, or branch [7], the coverage monitor gives a set of *requirements* $\mathcal{R}(T) = \{r_1, r_2, \dots, r_m\}$ for a test suite T . The prioritizers use the set of *covered* requirements for each test $\mathcal{R}(t_i) \subseteq \mathcal{R}(T)$ to *reorder the suite*.

Table 1 shows an example of a test suite with 4 tests and 5 requirements. An X in a cell (t_i, r_j) means that $r_j \in \mathcal{R}(t_i)$. 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 developers can more quickly begin to make corrections.

The coverage effectiveness (CE) metric rates a test suite order based on how fast it covers each requirement $r_j \in \mathcal{R}(T)$ [4]. The execution of each test suite offers the possibility of covering more total requirements and when the cumulative coverage is plotted against time a step function is formed as shown in Figure 1. CE is calculated by dividing the area under the step function for the actual order by the area under the curve of the ideal test suite that covers all of its requirements immediately. This value is inclusively between 0 and 1 where a CE of 0 means that the test suite did not

	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

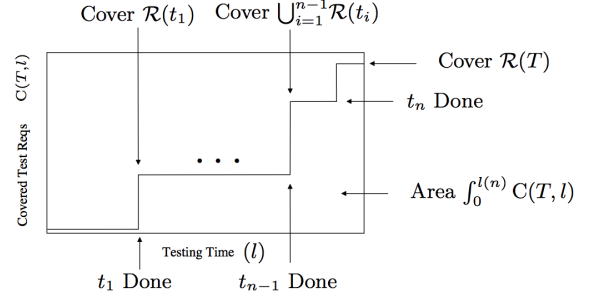


Figure 1: Coverage Effectiveness.

execute any requirements and a CE of 1 indicates that the test suite instantly covers all of the requirements.

For a test suite with n tests it is too expensive to find the best CE value by generating and evaluating all n possible orders. In response to this challenge this paper presents an interactive visualization technique with tool support that generates and plots the results of the greedy (GRD), 2-optimal greedy (2OPT), delayed greedy (DGR), and Harrold Gupta Soffa (HGS) test suite prioritization algorithms [6]. These prioritization techniques 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 orders that are favored over the original order [2]. Generating large random samples also gives insight into the distribution of CE values across many different orders of a test suite. The reverse order of a test suite may also produce higher CE values than the original and is included in this visualization technique [6].

It is difficult to interpret the results of several algorithms or random samples by reading text alone. Therefore, this paper presents a visualization technique that can allow software testers to view the cumulative coverage and CE of a large set of test suite orders simultaneously. Related work in the field of visualizing test suites exists [3], however, it focuses on fault localization or different features of a test suite. Examining the CE functions of several orders for a given test suite will quickly reveal the effectiveness of the different prioritization techniques on that test suite. Figure 1 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.

2 TOOL DESIGN

This visualization technique aids in analyzing multiple prioritizations of a test suite through interactive CE multiplots. Drawing from characteristics shown by Becker et al. [1] and a NY Times

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

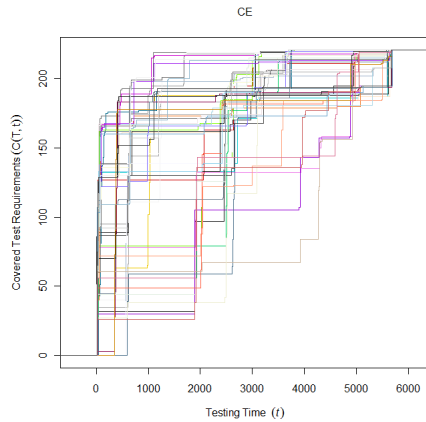


Figure 2: Static Coverage Effectiveness Multiplot

interactive visualization of market statistics¹ the tool allows users to interactively select techniques displayed and directly manipulate the corresponding plot. Data on demand obviates any need for keys or legends. 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 thin gray lines. Below the slider bar the average CE and standard deviation of CE are displayed for the current sample and also for the entire collection of random prioritizations sampled so far.

The right panel displays the multiplot. The y axis displays the number of requirements and the x axis shows the execution time. The step functions for the cumulative coverage of the chosen test orders are plotted in this area. In Figure 3 all techniques are currently selected to appear in the multiplot. A mouse-over on a function line highlights it and shades the area under it, as well as provides a label identifying the technique, greedy choice metric, and the CE value of the prioritization represented by the line.

3 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] H. Do, G. Rothermel, and A. Kinneer. Empirical studies of test case prioritization in a junit testing environment. In *ISSRE '04: Proceedings of the 15th International Symposium on Software Reliability Engineering*, pages 113–124, Washington, DC, USA, 2004. IEEE Computer Society.
- [3] 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.
- [4] G. M. Kapfhammer and M. L. Soffa. Using coverage effectiveness to evaluate test suite prioritizations. In *Proceedings of the ACM Work-*

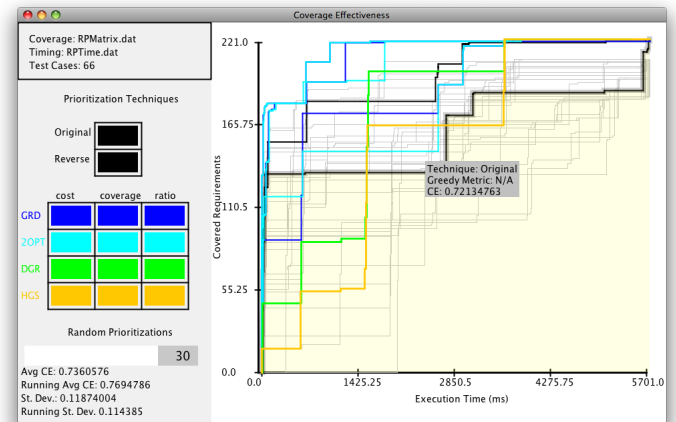


Figure 3: Interactive Coverage Effectiveness Multiplot

shop on *Empirical Assessment of Software Engineering Languages and Technologies*, November 2007.

- [5] 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.
- [6] 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.
- [7] 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.

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