An Empirical Comparison of Methods for Compressing Test Coverage Reports

Erik Ostrofsky and Gregory M. Kapfhammer
Allegheny College
Department of Computer Science
{ostrofe, gkapfham}@allegheny.edu

Introduction. Test coverage monitoring techniques are an integral part of modern methodologies for testing computer software. For instance, tools such as automated fault localizers [2], test adequacy calculators [3], and debuggers [4] all use a coverage report for various purposes. Recently developed monitoring methods track the coverage of the nodes and edges in a program's control flow graph, definition-use associations involving program variables [8], or interaction with the state and structure of a database [3]. However, a coverage report often balloons in size as the monitor includes additional details about the behavior of the program, test suite, and other software components such as the operating system and database. The marked increase in coverage report size is particularly problematic when testing occurs in a resource constrained embedded environment [1] or on a build/test server that collects coverage results for many programs over a long time period [7]. Large coverage reports may also limit the efficiency and effectiveness of defect isolation methods that monitor a remote program and thus transmit coverage data across a network [5].

Empirical Study. As shown in Figure 1, the monitoring process normally involves the following steps: (1) instrumenting the program and test suite in order to collect coverage information, (2) executing the test suite so that a coverage report may be constructed, and (3) compressing and storing the coverage results for later analysis. This poster studies the impact that coverage report format and the use of compression has on the third phase of monitoring. Since prior work finds that coverage reports in the form of either a dynamic call tree (DCT) or a calling context tree (CCT) support effective test suite reduction [3, 6], this poster picks this tree-based format as the basis for an empirical comparison of compression methods. In particular, we experimentally evaluate the trade-offs associated with encoding the coverage report in either a binary or eXtensible markup language (XML) file and further consider the merits of using either a traditional or XML-aware file compressor. Using six case study applications that vary in their size (548 to 1455 non-commented source statements and 13 to 51 test cases) [3], this poster reports on measurements of the time overhead associated with storing the report, size of both the compressed and uncompressed coverage results, and time required to run the compressor and decompressor.

Using graphs and tables like those in Figure 2, this poster enables both researchers and practitioners to identify the fundamental trade-offs associated with compressing coverage data. For instance, the experiments reveal that the reports encoded in XML take longer to store and are larger than the comparable binary file. Yet, an XML-aware compressor called XMill exploits the meta-data embedded in this text-based format in order to create a coverage file that is smaller than the compressed binary version.

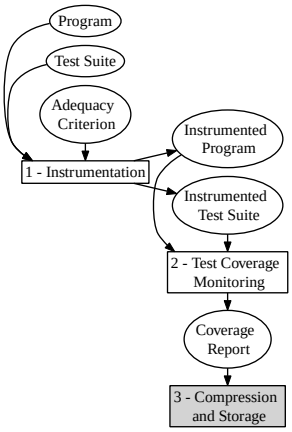

Figure 1. Test Coverage Monitoring Process.

Report	Format	Time (ms)
CCT	Binary	144.9
DCT	Binary	1011.72
CCT	XML	408.17
DCT	XML	2569.22

Format	Original Size (kb)
Binary	39.1
XML	283

Format	Compressor	Size (kb)
Binary	Gzip	3.59
Binary	Zip	3.94
XML	Gzip	6.73
XML	Zip	7.09
XML	XMill	2.36

Figure 2. Report Overheads Across All Applications.

References

[1] S. Bhadra, A. Conrad, C. Hurkes, B. Kirklin, and G. M. Kapfhammer. An empirical study of methods for executing test suites in memory constrained environments. In *Proc. of 4th AST*, 2009.

[2] J. A. Jones and M. J. Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proc. of 20th ASE*, 2005.

[3] G. M. Kapfhammer and M. L. Soffa. Database-aware test coverage monitoring. In *Proc. of the 1st ISEC*, 2008.

[4] A. J. Ko and B. A. Myers. Debugging reinvented: asking and answering why and why not questions about program behavior. In *Proc. of 30th ICSE*, 2008.

[5] B. Liblit, A. Aiken, A. X. Zheng, and M. I. Jordan. Bug isolation via remote program sampling. In *Proc. of PLDI*, 2003.

[6] S. McMaster and A. M. Memon. Call stack coverage for test suite reduction. In *Proc. 21st ICSM*, 2005.

[7] A. M. Memon, I. Banerjee, and A. Nagarajan. DART: A framework for regression testing nightly/daily builds of GUI applications. In *Proc. of 19th ICSM*, 2003.

[8] J. Misurda, J. A. Clause, J. L. Reed, B. R. Childers, and M. L. Soffa. Demand-driven structural testing with dynamic instrumentation. In *Proc. of 27th ICSE*, 2005.