# An Experimental Comparison of Four Test Suite Reduction Techniques

Hao Zhong, Lu Zhang[*], Hong Mei[*]
Institute of Software, School of Electronics Engineering and Computer Science, Peking University,
Beijing, 100871, P.R. China
86-10-62751794
{zhonghao04, zhanglu, meih }@sei.pku.edu.cn

## ABSTRACT

As a test suite usually contains redundancy, a subset of the test suite (representative set) may still satisfy all the test objectives. As the redundancy increases the cost of executing the test suite, many test suite reduction techniques have been brought out in spite of the NP-completeness of the general problem of finding the optimal representative set of the test suite. In the literature, some experimental studies of test suite reduction techniques have already been reported, but there are still shortcomings of the studies of these techniques. This paper presents an experimental comparison of the four typical test suite reduction techniques: heuristic H, heuristic GRE, genetic algorithm-based approach and ILP-based approach. The aim of the study is to provide a guideline for choosing the appropriate test suite reduction techniques.

## Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging, Debugging aids;
D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement

## General Terms

Algorithms, Measurement, Performance, Experimentation,

## Keywords

Test suite reduction, Software testing, Test suite minimization, Empirical studies.

## 1. INTRODUCTION

In real world software development, developers typically rely on testing to find bugs in the software. Usually, testing is an expensive process, and one key factor for the expensiveness of testing is that it typically takes a very long period of time to execute the whole set of test cases. Therefore, it has long been identified as a research focus to find a small but effective set of test cases for testing a software system.

[*] Corresponding Authors

As one test case can hardly satisfy all the requirements, it is usually required to use a suite of test cases to satisfy as many as possible requirements. Intuitively, the more test cases are used, the more possible the requirements are satisfied. Practically, a test suite usually undergoes a process of expansion, as new test cases are inserted into the test suite to ensure the requirements being satisfied. As a result, a test suite may contain more than enough test cases for satisfying the requirements. That is to say, when some test cases are removed from the test suite, the test suite may still satisfy all the requirements that can be satisfied by the original test suite. Thus, finding a sub-suite of an existing test suite that can satisfy the same requirements as the original test suite becomes a research problem. This problem is usually referred to as *test suite reduction* and the sub-suite of test cases is usually called the *representative set*. Obviously, test suite reduction can decrease the time of executing the test suite, and thus decrease the cost of testing.

In the previous research, many efforts have been put into research on how to reduce the size of a test suite while maintaining its effectiveness. Typical test suite reduction techniques can be found in [1], [2], [4] and [6] etc. Although there have been some experimental evaluation or comparison of some test suite reduction techniques (such as [3]), there are still some shortcomings of current experimental studies of these techniques: First, there is no comparison of recently proposed techniques, such as [1], [2], [6], only some of which are evaluated against [2] and [4]. Second, some experimental comparison is based on simulation data (see [3]), which may not reflect the same situation in reality. Third, most subject programs are very small (such as the Siemens programs [5]) and the results on small programs may not be applicable for large programs.

To address these shortcomings, we implemented four typical test suite reduction techniques on the same platform and performed an experimental comparison of them by applying them on both small and large subject programs. In this paper, we present the results of our experiment, and based on the analysis of these results, we also present some insights into the selection of test suite reduction techniques.

The organization of the remaining of this paper is as follows. The design of our experiment is reported in section 2. In section 3, we present the results and analysis of our experiment, and we conclude this paper in section 4.

## 2. THE EXPERIMENT

### 2.1 Implementation of the Techniques

Our experiment is performed on a PC with an INTEL Pentium IV CPU 2.26GHz and 512M memory. All the studied test suite reduction techniques are implemented on this PC by a single software engineer using VC++6.0 and all the executables run on Windows 2000 Professional. Heuristic H [4] and heuristic GRE [2] are simply implemented as C++ programs. The approach using the hybrid genetic algorithm (GA) [6] is implemented using Galib developed by Wall [10]. For the approach using ILP [1], IBM's SYMPHONY [7] is employed for solving the ILP model. As there are two ILP models proposed in [1] and only the first ILP model aims at achieving smallest representative sets, we only used the first ILP model in the implementation to make the ILP-based approach comparable to the other three techniques,.

### 2.2 Subject Programs and Test Cases

In our experiment, six C programs were used as subjects. The first four programs are from the well known Siemens program suite, which was originally provided by researchers at Siemens Corporate Research and can be downloaded from Aristotle analysis system's homepage [13]. We use both the source code and the test cases downloaded from this homepage in our experiment. The other two programs are XMLPPM and GNU Tar. XMLPPM is an XML compressor that can be obtained from [14]. In our experiment, we use a collection of XML files stored on the PC as test cases. GNU Tar is an archiver that creates and handles file archives in various formats. The source codes of GNU tar can also be obtained from [15]. In our experiment, we used a collection of files under one large directory on the PC as test cases.

### 2.3 Experimental Procedure

In our experiment, we use statement coverage as the requirements. As there are usually some none executable statements in a program and some executable statements may not be covered by any used test cases, the number of requirements for each program is typically smaller than its number of lines of code.

In order to extract the coverage information of test cases of the subject programs, we instrumented the source code, and used a specially developed script to run the instrumented executable with the test cases and create the profiling files in the **.gcov* format. Then we use GcovReader to collect and interpret all the profiling files to produce the test suite pool. The test suite pool is a two dimension matrix, in which, a column stands for a requirement and a row stands for a test case. If a test case can satisfy a requirement, the corresponding position in the test suite pool will be marked as 1, otherwise marked as 0. During the procedure, a coarse reduction is done to remove the test cases that satisfy the exactly the same requirements satisfied by another test case and the requirements that cannot be satisfied by any test case. The information of these subject programs and their test suite pools are listed in Table 1.

**Table 1. Subject programs and their test suite pools**

| Programs | Source File Size (LOC) | Test Suite Pool Size (T×R) |
|---|---|---|
| print_tokens | 447 | 3970×128 |
| Replace | 512 | 4068×259 |
| Schedule | 282 | 2287×157 |
| Tcas | 135 | 719×70 |
| Xmlppm | 3251 | 1694×983 |
| Tar | 26824 | 611×2403 |

To study the ability of dealing with test suites of different sizes for the four test suite reduction techniques, we used a test suite selector to randomly choose a subset of test cases from the test suite pool for each subject program to form a test suite. During the procedure, another coarse reduction was done to remove the requirements that cannot be satisfied by any of the selected test cases. This step is essential since all the four test suite reduction techniques require that all the requirements be satisfied by at least one test case in the test suite. This test suite was then sent to the implementation of the four different test suite reduction algorithms. After the execution, a log file was produced to record the execution time and the representative set for each test suite reduction technique. Our comparison was based on the information recorded in all the log files.

## 3. Results and Analysis

The central features of the four techniques studied in our experiment are 1) whether the four techniques can be effective in reducing the size of the test suite; and 2) whether the four techniques can perform their tasks in acceptable time. Therefore, our comparison concentrated on the following things:

### 3.1 Representative Sets

The sizes of the representative sets are depicted in Fig. 1, from which, we can see that these algorithms produce almost the same sizes of representative sets except the approach using the hybrid genetic algorithm, although all the four techniques can significantly reduce the sizes of test suites. Actually, the genetic algorithm-based approach can produce good representative sets as other approaches for some subjects, such as Tcas, but it loses in most cases especially when complicacy becomes high.

To make the comparison of the other three techniques clearer, we use only the data of these three approaches to form Fig. 2. In this figure, these three approaches are still inseparable for many cases. In average, the difference between the three techniques is less than 1%.

When we closely examine this figure, we find that the ILP-based approach can always produce the smallest representative set for every situation. For heuristic GRE and heuristic H, no one can guarantee which is more advantageous. Actually, besides the circumstances that GRE and H produce the same sizes of representative sets, each one wins the other for about 50% of the rest circumstances.

## 3.2 Execution Time

Fig. 3 depicts the relationship between the complicacy of the test suites and the execution times. The complicacy of the test suite pool is calculated as $log_{10}(mn)$ where $m$ stands for the number of requirements that the representative set should satisfy and $n$ stands for the number of test cases in the test suite. The complicacy is not continuous as the test suites are randomly selected from the test suite pool.

Generally speaking, with increase of the complicacy of the selected test suite, all these algorithms will consume more time, but it takes much longer for the genetic algorithm-based approach to produce representative sets than other approaches.

As the other three techniques become inseparable in this figure, we produce Fig. 4 only using the data of these three approaches. From both figures, we can see that heuristic H needs the least time to calculate the representative set, while heuristic GRE and the ILP-based approach are about the same. Actually, GRE are a little faster in most circumstances, but the difference can hardly have big impact on testing, as it is not comparable to the typical time of executing test cases. Thus, the time efficiency of these four algorithms can be summarized as $t_{GA} >> t_{ILP} \approx t_{GRE} > t_{H.}$

## 4. CONCLUSIONS AND FUTURE WORK

In this paper, we performed an empirical comparison of four test suite reduction techniques: heuristic H, heuristic GRE, the genetic algorithm-based approach and the ILP-based approach. The comparison targets at two main factors of test suite reductions: representative sets and execution time. Findings of this comparison can be summarized as follows:

- All the four techniques can dramatically reduce the sizes of test suites. Except the genetic algorithm-based approach, which performs much weaker in this aspect, the other three can produce almost the same sizes of representative sets.
- The ILP-based approach can always produce the smallest representative sets among all the four approaches. For Heuristic H and heuristic GRE, it is hard to tell which one is superior to the other. This is a little different from the result in [3], in which, GRE always plays better than H. We think the reason lies in that our comparison is based real data while [3] is based on simulation data.
- The genetic algorithm-based approach also performs the worst in the aspect of execution time. For the other three, all of them can produce representative sets in acceptable time. Among them, heuristic H is the fastest, while heuristic GRE is a little faster than the ILP-based approach. Considering their ability to produce small representative sets, we suggest that heuristic H should be the first choice and the ILP-based approach should be preferable when the smallest representative sets are required.

In the literature, it is still a controversy whether test suite reduction can decrease the ability of the test suite to detect faults. In [9], Thevenod-Fosse et al. reported a case study in which redundancy can increase the fault detecting ability of the test suite. In [11] and [12], Wong et al. showed that the representative sets have almost the same capability to reveal bugs as the original test suites. In a recent experimental study, Rothermel et al. found that the fault-detection capabilities of test suites can be severely compromised by test suite reduction [8]. In our experiment, we find that different techniques can produce very different representative sets for the same circumstances, although the sizes of the representative sets are about the same. In the future, we will investigate whether and in what circumstances a particular test suite reduction technique can produce representative sets with high fault-detection ability.

## 5. ACKNOWLEDGMENTS

This effort is sponsored by the National 973 Key Basic Research and Development Program No. 2002CB312003, the State 863 High-Tech Program No. 2005AA113030, and the National Science Foundation of China No. 60403015.

## 6. REFERENCES

[1] J. Black, E Melachrinoudis, and D. Kaeli. “Bi-Criteria Models for All-Uses Test Suite Reduction,” International Conference on Software Engineering, 2004, pp. 106-115.

[2] T.Y. Chen and M.F. Lau, “A New Heuristic for Test Suite Reduction,” Information and Software Technology, Vol. 40, No. 5, 1998, pp. 347-354.

[3] T.Y. Chen and M.F. Lau, “A Simulation Study on Some Heuristics for Test Suite Reduction,” Information and Software Technology, Vol. 40, No. 13, 1998, pp. 777–787.

[4] M.J. Harrold, R. Gupta, and M.L. Soffa. “A Methodology for Controlling the Size of a Test Suite,” ACM Transactions on Software Engineering and Methodology, Vol. 2, No.3, 1993, pp. 270-285.

[5] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, “Experiments on the Effectiveness of Dataflow- and Control Flow-Based Test Adequacy Criteria,” International Conference on Software Engineering, 1994, pp. 191-200.

[6] N. Mansour and K. El-Fakin. “Simulated Annealing and Genetic Algorithms for Optimal Regression Testing,”

Journal of Software Maintenance: Research and Practice, Vol. 11, No. 1, 1999, pp. 19–34.

[7] T. Ralphs and M. Guzelsoy, "The SYMPHONY Callable Library for Mixed Integer Programming," The Ninth INFORMS Computing Society Conference, 2005, pp. 61-73.

[8] G. Rothermel, M.J. Harrold, J. von Ronne, and C. Hong. "Empirical Studies of Test-Suite Reduction," Software Testing Verification and Reliability, Vol. 12, No. 4, 2002, pp. 219-249.

[9] P. Thevenod-Fosse, H. Waeselynck, and Y. Crouzet, "An Experimental Study on Software Structural Testing: Deterministic verses Random Input Generation," IEEE International Symposium on Fault Tolerant Computing, 1991, pp. 410-417.

[10] M. B. Wall, *A Genetic Algorithm for Resource-Constrained Scheduling*. MIT, PhD thesis, 1996.

[11] W.E. Wong, J.R. Horgan, S. London, and A.P. Mathur, "Effect of Test Set Minimization on Fault Detection Effectiveness," Proc. of the 17th International Conference on Software Engineering, 1995, pp. 41–50.

[12] W.E. Wong, J.R. Horgan, A.P. Mathur, and A. Pasquini, "Test Set Size Minimization and Fault Detection Effectiveness: A Case Study in a Space Application," Annual International Computer Software and Applications Conference (COMPSAC), 1997, pp. 522–528.

[13] http://www.cc.gatech.edu/aristotle/Tools/subjects/

[14] http://xmlppm.sourceforge.net/

[15] http://www.gnu.org/software/tar/

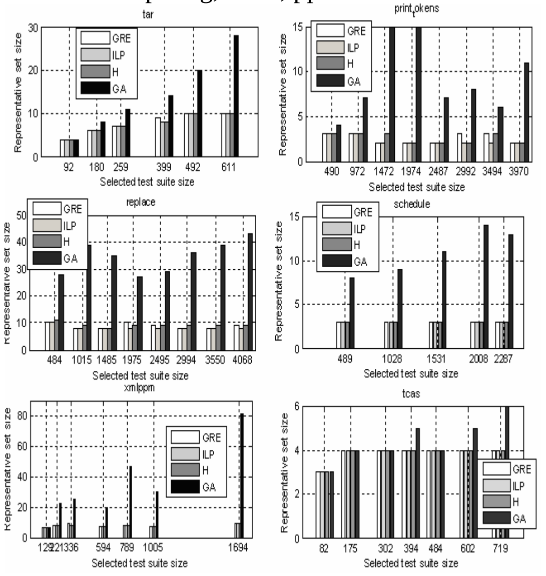

**Fig.1.Sizes of representative sets for GRE, ILP, H and GA**

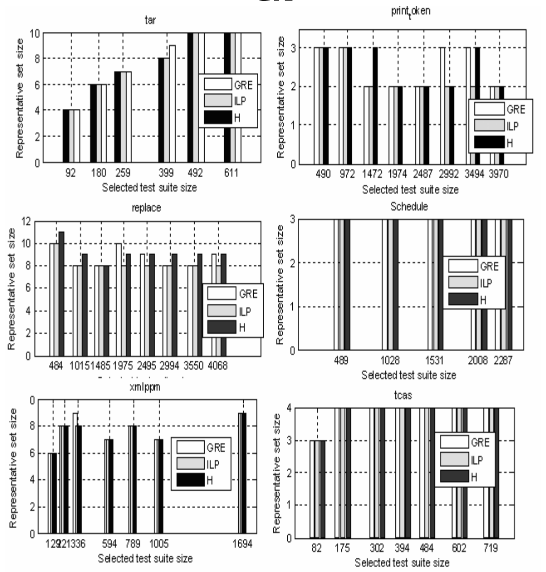

**Fig. 2. Sizes of representative sets for GRE, ILP and H**

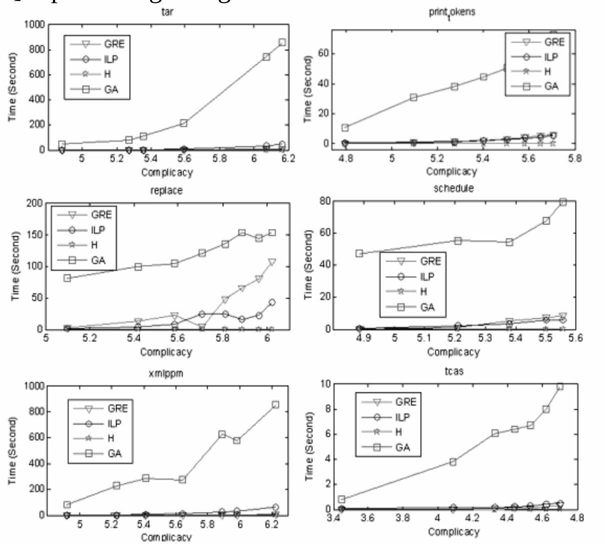

**Fig. 3. Execution times of GRE, ILP, H and GA**

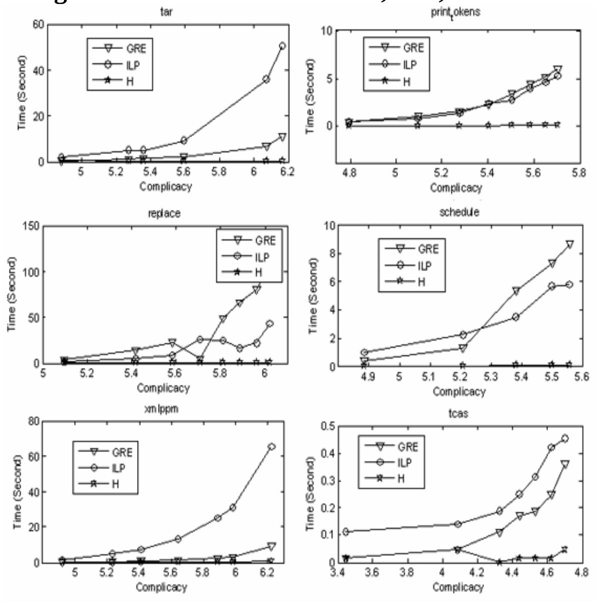

**Fig. 4. Execution times of GRE, ILP and H**