Evaluating the Use of a General-Purpose Benchmark Suite for Domain-Specific SMT-solving

Andrew Healy[*]
Dept of Computer Science
Maynooth University
Co. Kildare, Ireland
ahealy@cs.nuim.ie

Rosemary Monahan
Dept of Computer Science
Maynooth University
Co. Kildare, Ireland
rosemary@cs.nuim.ie

James F. Power
Dept of Computer Science
Maynooth University
Co. Kildare, Ireland
jpower@cs.nuim.ie

ABSTRACT

Benchmark suites are an important resource in validating performance requirements for software. However, general-purpose suites may be unsuitable for domain-specific purposes, and may provide an incorrect indication of the software performance.

This paper uses SMT-solvers (*Satisfiability Modulo Theories*) as a case-study. Taking deductive software verification as a specific application domain for SMT-solvers, we present an approach to quantifying the difference between general-purpose and domain-specific benchmark suites. We show that workload-based clustering of benchmark programs increases the specificity of features tested by the suite compared to the inherent hierarchy of a general-purpose suite.

CCS Concepts

•**Software and its engineering** → **Software performance;** *Software verification;*

Keywords

Benchmarks; SMT solvers; software verification, profiling

1. INTRODUCTION

The use of benchmarking can provide useful indicators to characterise the workload of a software system when completing a specific task and is an important element of non-functional testing such as software performance engineering.

[*]This project is being carried out with funding provided by *Science Foundation Ireland* under grant number 11.RFP.1/CMS/3068

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

SAC 2016,April 04-08, 2016, Pisa, Italy

©2016 ACM. ISBN 978-1-4503-3739-7/16/04...$15.00

DOI: http://dx.doi.org/10.1145/2851613.2851975

Ideally, the design and analysis of such suites should be subject to the same rigour as other artefacts of the software testing cycle [7].

This paper addresses the construction and analysis of a *domain specific* benchmark suite. The context of our work is that we have access to a very large general purpose benchmark suite (SMT-LIB for SMT solvers), but we wish to benchmark programs in a particular sub-domain of application (software verification). We have assembled a suite of programs to act as a benchmark suite for this domain-specific purpose, and we wish to be able to quantify any difference in the workload characteristics between our domain-specific suite and the general-purpose suite.

SMT-solvers extend SAT-solvers by combining specialised decision procedures for a range of logical theories; for example, the Z3 SMT-solver has specialised theories for bit-vectors, arithmetic, quantifiers, uninterpreted functions, arrays and other datatypes. The development of SMT-solving tools has seen significant advances in the past decade due, in part, to the widespread utility of the tools in software engineering. SMT solvers are used in operations research, cryptology, machine learning and formal verification. The active SMT-LIB community organises tool competitions, oversees the development of a common input language, and collects the benchmark programs that provides the context for this paper.

SMT-solvers are increasingly used in deductive software verification, but there is no standardised benchmark suite to compare such verification systems. There is a wide variety of specification and annotation languages for programmers to choose from when declaring contracts and invariants for their programs. This variety makes the comparative evaluation of deductive software verification systems difficult [2].

2. EXPERIMENTAL SETUP

This section outlines the sequence of steps we took to select the data and tools used in our experiments, and defines the metrics used to distinguish between benchmark suites.

2.1 Selecting benchmark programs

In order to assemble a domain-specific benchmark suite for software verification we made use of the examples in the **Why3** distribution [4]. Our selection of verification programs from the Why3 distribution was limited to programs that, as much as possible, did not require the use of interac-

tive theorem provers such as *Coq* or *Isabelle* to verify fully. We assembled a total of 116 of these verification programs, which we refer to from now on as the *verification (benchmark) suite*.

Currently, the SMT-LIB benchmark repository consists of over 100,000 programs. At the highest level, the repository is organised into directories according to the logics which must be supported by the prover in order to solve the problems in that directory. We use these categories to partition the suite in subsection 3.1.

Due to the high cost of instrumentation, for this study we took 50 random samples from the entire SMT-LIB suite, with each sample consisting of 116 programs (the same size as the verification suite). This process resulted in a selection of 5800 unique programs. We believe this sample is representative of the workloads tested by the entire repository due to the high degree of redundancy present in the entire suite. Added to our selection of verification programs, our suites consisted of 5916 programs in total.

The SMT provers had to meet particular criteria: in order to make use of the full range of programs included in the SMT-LIB repository, the provers must implement as many logics as possible. For the same reason, the tools were required to be fully compliant to the second version of the SMT-LIB language standard. Two SMT provers were found to meet these conditions: **Z3** [5] and **CVC4** [1]. Both these tools are used by software verification systems in addition to Why3; e.g. *Boogie, Dafny, Spec#* (Z3), and *Cascade, Stardust, GPUVerify* (CVC4).

2.2 Feature Extraction and Profiling Method

One obvious way to compare the suites is to run the SMT solvers and measure the time taken to process each suite. However, this would bias our survey to the features of the particular architecture used in the experiments, such as operating system, memory, disk and cache performance etc. Instead we aimed to create a platform independent evaluation that would not be affected by such details. To this end, we regarded each of the SMT solvers as if it were a virtual machine, with the method calls acting as virtual machine instructions. The result was a data set for the workloads that were machine- and platform-independent.

To conduct the analysis we used the *Callgrind* tool from the *Valgrind* project. For each program in the benchmark suite we recorded the number of times each method of the target SMT solver was called, along with the total number of method calls for the solver.

The total number of unique methods called (after filtering out system calls) leads to very large feature vectors. To avoid the "curse of dimensionality" [3], the number of features was reduced using Principal Component Analysis (PCA), where the number of dimensions remaining was chosen using a maximum likelihood estimation method [9]. This process increased the effectiveness of our distance measurement and the K-means clustering algorithm employed in Experiment 2. When viewing the results, it should be borne in mind that PCA significantly reduced the suites' difference measurement while maintaining their *relative* differences.

2.3 Metrics for benchmark suite design

The results discussed in Section 3 are characterised in terms of a *difference metric* for benchmark programs, and a *utilisation metric* for benchmark suites. This work follows the methodology outlined by Dujmović [7].

2.3.1 Measuring workload difference

We characterise the workload for a given benchmark program and solver as a feature vector of the proportion of total calls to each method. We normalise the count of calls to a proportional cost value, so that all costs sum to 1.

As the non-overlapping individual values for each method of the program are known, we can use these values to find the "white-box" difference between two programs A and B:

$$d(A,B) = \frac{1}{2}\sum_{i=1}^{n}|c_i^{(A)} - c_i^{(B)}|$$

Here n is the number of methods, c_i is the proportion of method calls to the ith method, and $d(A,B)$ is a variation on the Manhattan distance metric. The Manhattan or "city block" method is preferred to the Euclidean distance because the high number of zero-values in many dimensions would distort the metric.

For all benchmark programs A and B we have $0 \leq d(A,B) \leq 1$, with $d(A,B)$ giving a value of 0 for entirely similar workloads and 1 for those that are entirely different.

2.3.2 Measuring a suite's utilisation

Calculating the distance metric $d(A,B)$ for any two programs in the benchmark suite allows us to construct a 5916×5916 distance matrix for each SMT solver. For each experiment, we then treat the verification suite and each subset of the SMT-LIB as an individual suite. For each suite we used the distance metric to calculate a *centroid* program profile as the best representative of the suite.

The centroid was determined by first finding the *maximum* distance from each program to any other in the suite and then selecting the benchmark program for which this value was *minimal*. We can then find the distance between two *suites* by using their centroid programs in the white-box difference formula.

Next we determined the size D and the maximum size D_{max} for each suite (again following Dujmović [7]):

- D is the diameter of the minimum hypersphere that contains all programs in the suite, and is calculated as twice the distance from the centroid to the furthermost program from the centroid.
- D_{max} is the maximum distance between a program and every other program or group of programs.

Given the size D and the maximum size D_{max} for each suite, we can now calculate the suite's *utilisation of the program space* as a percentage: $100 * D/D_{max}$. Suites with low utilisation display a high degree of redundancy - i.e. many benchmarks exercise the same features. Another way of interpreting this figure is the "tightness" of the clustering around the centroid.

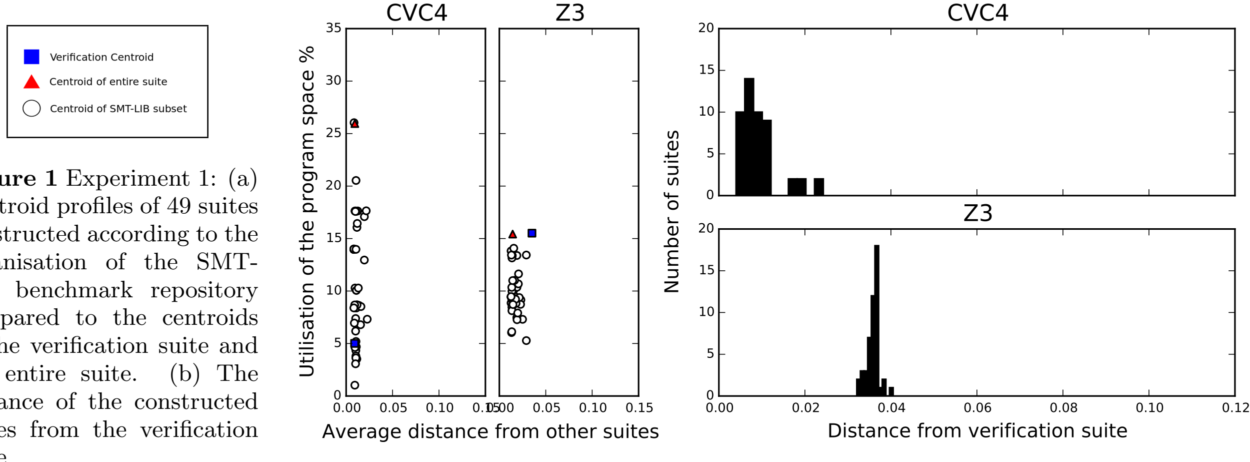

Figure 1 Experiment 1: (a) Centroid profiles of 49 suites constructed according to the organisation of the SMT-LIB benchmark repository compared to the centroids of the verification suite and the entire suite. (b) The distance of the constructed suites from the verification suite

3. EXPERIMENTAL RESULTS

Two experiments were devised to answer the questions:

RQ1: Guided by the given structure of the general-purpose suite, does the verification suite bear a particular similarity to any branch of this hierarchy?

RQ2: Can we reconfigure the general-purpose suite using workload-based clustering to identify a suite with a similar profile to our verification suite?

Data related to this work is available from our website.[1].

The left side of Figures 1 and 2 contain a plot for each of the two SMT solvers, where each dot represents a benchmark suite (actually, the centroid of the suite). The positions of each verification suite and the centroid of all suites combined are marked using a square and triangle respectively. The horizontal axis measures the white-box distance between a suite and the others (on average), and is thus a metric of the distinctiveness of a given suite. Distance along the vertical axis measures the utilisation for this suite, and gives a metric for how varied the suite is in terms of workload, so lower values indicate a more specialised suite. Note that the utilisation figures for the entire suite and verification suite are invariant for both experiments.

The right-side of each figure shows a histogram that depicts the distances between the verification suite and each of the other suites in the experiment. The height of the bar (vertical axis) shows the number of benchmark suites at that distance (horizontal axis) from the verification suite.

3.1 RQ1: Verification suite in relation to an inherent hierarchy

If a large general-purpose benchmark suite is to be useful as a resource to aid the development of domain-specific tools, it should be structured in such a way that makes it clear which benchmarks are relevant for the tool's application domain. Experiment 1 intends to show whether any branch of the SMT-LIB benchmark repository's structure shows any correspondence to the workload for the verification suite.

[1]http://www.cs.nuim.ie/~ahealy/SAC2016

First, the directed acyclic graph of logic relationships was converted into a hierarchical tree structure by duplicating shared child nodes. Then suites of roughly the same size as our collection of verification programs were created from this new hierarchy through a simple greedy partitioning algorithm. The metrics described in section 2.3 were then obtained and plotted.

Figure 1(b) shows that distances from the verification suite are generally very similar to each other. Taking the utilisation results from Experiment 2 as a guide, we see that the inherent hierarchy represents a closer indication of workload for CVC4 than it does for Z3.

For CVC4, the closest constructed clusters to the verification workload were those from branches containing Quantifier-Free Linear Integer and Real Arithmetic (QF_LIA, QF_LRA) problems. For Z3, the closest clusters consisted of problems that required the use of many logics (AUFLIRA). This may account for the relatively high utilisation percentage for the Z3 verification suite. We deduce that more decision procedures are called during a software verification session on Z3 than CVC4.

3.2 RQ2: Re-clustering based on workload

Experiment 2 set out to determine whether we could repartition the general-purpose suite using behaviour-driven clustering as an alternative to the somewhat arbitrary divisions used in Experiment 1. We applied the K-Means clustering algorithm which is widely used and very efficient on large datasets. We choose 40 for the number of clusters in order to make a comparison with the previous experiment.

In Figure 2, the clustered suites in both Z3 and CVC4 display a marked increase in average distance from each other. This is an expected outcome of a workload-based clustering method. The general gathering of suites towards the lower end of the utilisation scale indicates that as the workload of the suite becomes more specialised, less of the program is being used. Again, relatively high utilisation for Z3's verification suite indicates that this suite does not represent a coherent workload cluster.

The number of suites to the right of Figure 2(b)'s horizontal

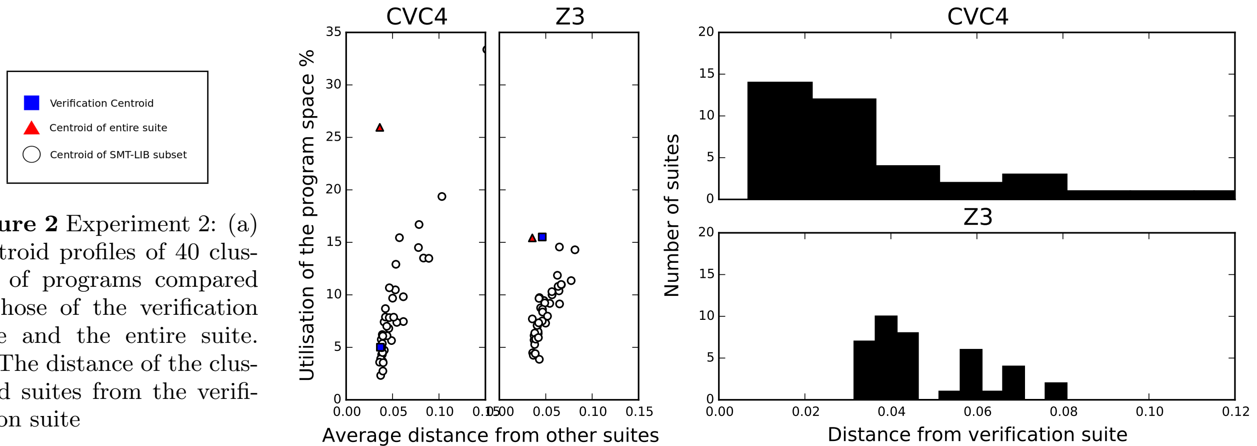

Figure 2 Experiment 2: (a) Centroid profiles of 40 clusters of programs compared to those of the verification suite and the entire suite. (b) The distance of the clustered suites from the verification suite

axis indicates that workload-based clustering can identify groups of programs that are significantly different to a verification workload. The technique did not identify a cluster that is closer to the verification workload than those found in the previous experiment, however.

3.3 Related Work

Demyanova et al. have used static profiling to calculate program metrics for verification problems [6]. Since these are used to classify programs, their work differs from ours, since we evaluate a suite of programs by dynamically measuring a solver's performance.

Sherwood et al. present a method of combining a basic-block sampling-based profiling technique with K-means cluster analysis [10]. They use these methods in order to gain an understanding of the behaviour of large programs across computer architectures.

Our work has some similarities with that of Jones and Harrold [8], which sought to reduce the size of large regression test suites using coverage criteria. However, their coverage criterion is not directly comparable to the utilisation percentages shown in Figures 1a and 2a.

4. CONCLUSION

This paper has presented a systematic quantitative method for the task-driven analysis and evaluation of a general-purpose benchmark suite for domain-specific purposes. We have used a platform-independent model to characterise what constitutes a program's *workload*, and concepts from cluster analysis to help classify a large body of programs. We set out to determine if a subset of the SMT-LIB repository could be identified that could then be used as a surrogate for a standard workload in deductive software verification.

Although we may not have found such a subset, we have shown that workload-based clustering can be used to eliminate programs irrelevant to this task. Such knowledge could be particularly useful in the case of high-cost benchmarking. We have also shown that following the SMT-LIB benchmark directory structure yields more cohesive workloads for CVC4 than it does for Z3, but that Z3 makes use of a wide range of its features when solving problems in the software verification domain.

5. REFERENCES

[1] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. CVC4. In *Computer Aided Verification*, volume 6806 of *LNCS*, pages 171–177. Springer, 2011.

[2] D. Beyer, M. Huisman, V. Klebanov, and R. Monahan. Evaluating Software Verification Systems: Benchmarks and Competitions (Dagstuhl Reports 14171). *Dagstuhl Reports*, 4(4):1–19, 2014.

[3] C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[4] F. Bobot, J.-C. Filliâtre, C. Marché, and A. Paskevich. Why3: Shepherd your herd of provers. In *Workshop on Intermediate Verification Languages*, pages 53–64, 2011.

[5] L. De Moura and N. Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[6] Y. Demyanova, T. Pani, H. Veith, and F. Zuleger. Empirical software metrics for benchmarking of verification tools. In *Computer Aided Verification*, volume 9206 of *LNCS*, pages 561–579. Springer, 2015.

[7] J. Dujmović. Universal benchmark suites - a quantitative approach to benchmark design. In R. Eigenmann, editor, *Performance Evaluation and Benchmarking with Realistic Applications*, pages 257–287. MIT Press, 2001.

[8] J. A. Jones and M. J. Harrold. Test-suite reduction and prioritization for modified condition/decision coverage. *IEEE Transactions on Software Engineering*, 29(3):195–209, 2003.

[9] T. P. Minka. Automatic choice of dimensionality for PCA. In *Advances in Neural Information Processing Systems*, pages 598–604. MIT Press, 2001.

[10] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. *SIGOPS Oper. Syst. Rev.*, 36(5):45–57, Oct. 2002.