Exploring Predictability of SAT/SMT Solvers

Robert Brummayer
Johannes Kepler University
Linz, Austria
robert.brummayer@jku.at
, Duckki Oe
The University of Iowa
Iowa City, Iowa, USA
duckki-oe@uiowa.edu
and Aaron Stump
The University of Iowa
Iowa City, Iowa, USA
astump@acm.org

Abstract

This paper seeks to explore the predictability of SAT and SMT solvers in response to different kinds of changes to benchmarks. We consider both semantics-preserving and possibly semantics-modifying transformations, and provide preliminary data about solver predictability. We also propose carrying learned theory lemmas over from an original run to runs on similar benchmarks, and show the benefits of this idea as a heuristic for improving predictability of SMT solvers.

1 Motivation

SAT and SMT (Satisfiability Modulo Theories) solvers have enjoyed tremendous performance improvements in the past ten years, increasing the automated-reasoning power available for applications like algorithmic verification, combinatorial design, planning, and others (e.g., [7, 5, 4]). Most work in the field has focused just on performance-oriented quality metrics for solvers. For example, the basic measure used in both the most recent (at the time of writing) SAT Competition and SMT Competition was simply the pair of the number of benchmarks solved and running time to solve them, compared in the natural lexicographic order (for the competitions mentioned, see, e.g., [1, 3]). While the SAT competition has also experimented recently with more complex measures, they are also centered on performance.

In this paper, we propose another property to consider when evaluating solvers, namely **predictability**. While users certainly require and benefit from improvements to raw performance, anecdotal evidence from end users of solvers suggests that in some cases unpredictability is at least as significant a concern. For example, Steve Miller, Principal Software Engineer in the Advanced Technology Center of Rockwell Collins, reported in his keynote presentation at Midwest Verification Day 2009 that unpredictability is a significant issue for his team in incorporating SAT/SMT solvers into their verification workflow. Unpredictability is a problem because a small change to a model can lead to an enormous change in the amount of time to solve the resulting verification condition. If the amount of time is enormously longer, the verification may become infeasible or unacceptably delayed. If it is enormously shorter, engineers may doubt the result, questioning if an error elsewhere in the workflow has led to such different system behavior. It may improve the usability of such solvers to sacrifice a modest amount of performance for improved predictability.

In this paper we provide a preliminary study of predictability of SAT (Section 2) and SMT (Section 3) solvers under small mutations of standard benchmarks. We use the standard deviation of solver times on a collection of mutants as a measure of variability. In the case of SMT solvers, we also propose and study a technique for heuristically improving predictability, by carrying over a selection of learned theory lemmas from the original run of the solver to the runs on the mutants.

2 Experiments with SAT Benchmarks

We survey how small changes to benchmarks affect the performance of SAT solvers. In particular, we evaluate the effects of semantics-preserving changes such as variable renaming and literal/clause reordering. Moreover, changes that may change the satisfiability status, e.g. adding resp. dropping arbitrary literals, are evaluated. The goal is to quantify the variability of solving times and to identify the effects of different types of variations.

2.1 Methods

From the SAT competition 2009, 5 solvers and 13 benchmarks were chosen. The solvers are some of the highly ranked ones in each category, and the benchmarks are of easy to medium difficulty that could be solved in about 300 seconds from the industrial category of the competition. Those solvers and benchmarks are listed below with solving times:

Benchmark	`CirCUs`	`lysat`	`MiniSat`	`mxc-sat09`	`precosat`
`ACG-10-5p0`	44.72	39.99	21.52	47.26	35.59
`AProVE09-07`	7.08	12.25	4.72	7.06	2.89
`AProVE09-17`	13.44	9.62	7.33	6.75	4.95
`AProVE09-20`	292.43	45.25	16.64	28.08	35.14
`countbitsrotate016`	27.94	61.81	24.2	11.7	14.5
`gss-14-s100`	65.2	22.79	18.37	13.54	47.97
`gus-md5-04`	4.92	5.62	2.17	16.91	5.3
`icbrt1_32`	14.59	15.29	7.72	11.61	11.34
`minand128`	9.86	49.8	20.01	55.83	13.77
`minandmaxor032`	7.73	6.23	3.65	5.51	4.92
`minxorminand032`	5.2	8.37	4.6	9.33	7.25
`minxorminand064`	106.21	157.31	106.16	327.19	151.4
`post-c32s-gcdm16-22`	249.57	115.06	97.41	151.21	51.15

Types of variations We made random changes to the original in order to simulate situations in which users query similar, but not identical, formulas repeatedly. Seven types of variations were performed and summarized in Figure 1. The first five variations preserve the semantics of the original formula and do not change the satisfiability status of the formula. In contrast, the variations *nlcx* and *nlca* may change the satisfiability status. For the variation *nlcx*, 0.01% of all non-unary clauses are considered small and an arbitrary literal (of the existing variables) replaces one random literal of each affected clause. The variation *nlca* performs even more changes. In particular, the same number of binary and ternary clauses are changed with probability 0.01%. One literal is added to each of those binary clauses and one literal is dropped from each of those ternary clauses. Note that we tried to avoid some of the possible unrealistic changes. Unary clauses, the literal-clause ratio, and the lengths of clauses are left unchanged in order to keep the inherent difficulty level of the formula.

Measure of Predictability For each type of variation and each benchmark, a random sample of 50 variations was generated and the solving times were recorded. Each solver has its own performance distribution over the same sample. For predictability, we only care about the spread of distribution or the variability of data. If the distribution of a solver is "narrower", we can say the solver is more predictable. To statistically quantify variability, we used the standard deviation of solving times. Obviously, a "small" standard deviation indicates high predictability.

Type	Description
l	literals in each clause are reordered
c	clauses of the formula are reordered
n	variable names are changed
lc	a combination of *l* and *c* variations
nlc	a combination of *n*, *l* and *c* variations
nlcx	in addition to *nlc*, one literal of non-unary clause is changed (0.01% of chance)
nlca	in addition to *nlc*, one literal is dropped from or added to clause (0.01% of chance)

Figure 1: Types of variations

2.2 Empirical Results

Figure 2 summarizes the variabilities of solving times induced by the different types of variation. For each variation and each solver, the standard deviations of solving times for all benchmarks were collected. Each bar in the graphs summarizes the distribution of the 13 standard deviations for a given solver and a given variation. The gray box of each bar represents the range of the middle half values, which are considered typical values. The line in the middle of box marks the median value and the "+" sign marks the average. The *whiskers* sticking out of box extend to adjacent values that are not farther away from the edge of the box than 1.5 times the height of the box. Small squares are values farther out than the adjacent values and considered outliers.

The result for the variation *l* shows that all solvers have small variability compared to those under other types of variations. Note that the scale of the graph is smaller than the others. Interestingly, reordering literals affects the predictability of `precosat` more than other solvers. This could be easily avoided by sorting the literals inside the SAT solver. The outstandingly high value of `CirCUs` is for `AProVE09-20`. Notably, the solver showed less predictability over all variations of that particular benchmark. The results for the other semantics-preserving variations are almost the same. All solvers showed very similar behavior in general, except a few outliers. The other notable outlier, which belongs to `mxc-sat09`, is `minxorminand064`. Our experimental results suggest that any single type of variation, except for *l*, is sufficient to shuffle the performance of solvers without changing semantics. More experiments are necessary to generalize this observation to other benchmarks.

The *nlcx* variation changed the variabilities for some benchmarks so that more outliers appear in the graphs. Interestingly, the highest outliers are all for `post-c32s-gcdm16-22`. However, the majority of benchmarks did not change the variability of solving times compared to the result for the variation type *nlc*. Interestingly, the *nlca* variation uniformly made the formula easier to solve and the solving times less variable across all solvers. At the same time, relative variability among solvers did not change much. Therefore, the graph looks similar to that of *nlcx* even if the scale of graph is different. Considering this variation, the highest outliers are all for `AProVE09-20`.

3 Experiments with SMT Benchmarks and Theory Lemmas

In this section, we explore mutation of SMT benchmarks, taken from the SMT-LIB library [2]. The following mutations are applied below with equal probability: (1) change the value of a real

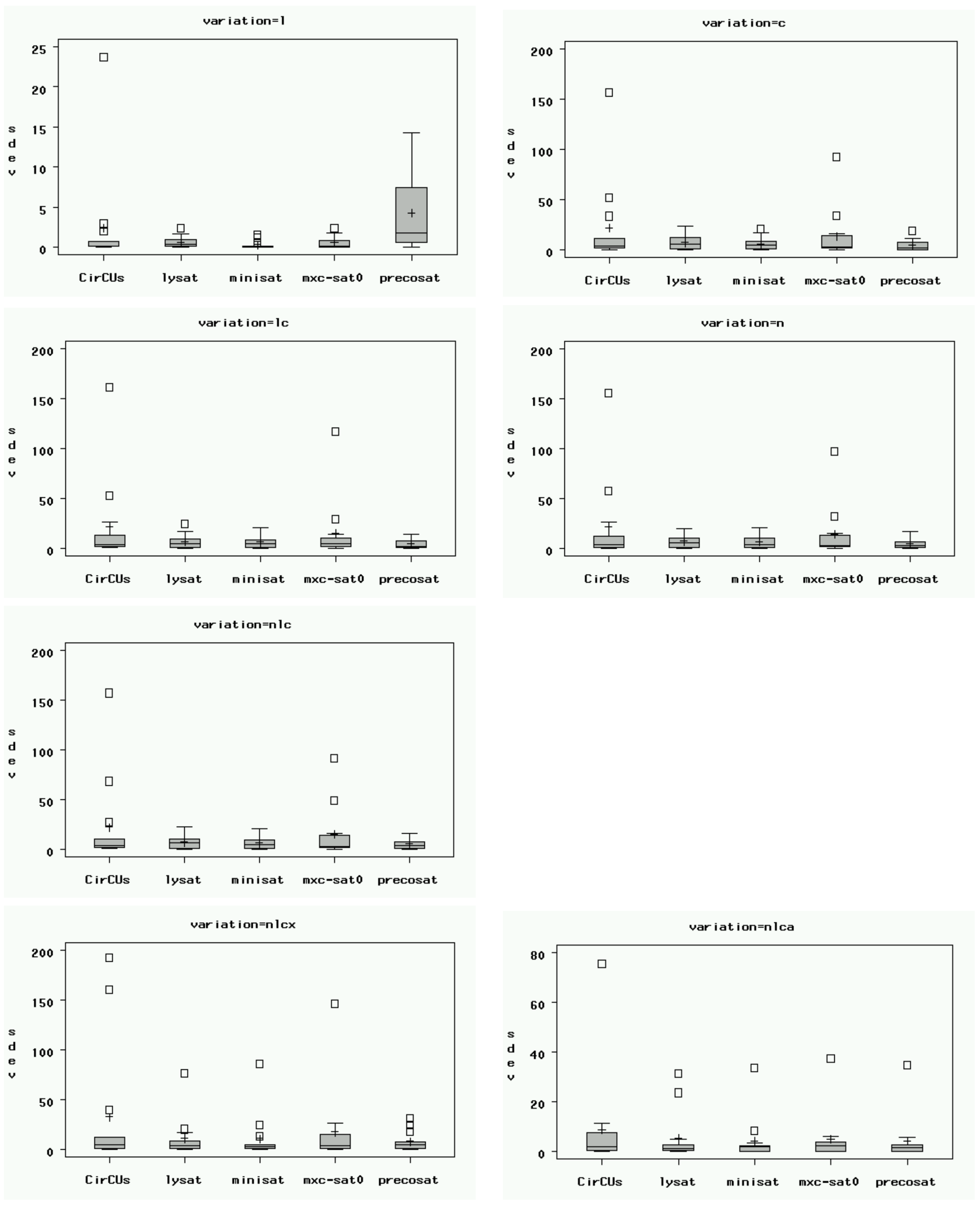

Figure 2: The variabilities of solving times by different types of variation

or integer constant; (2) swap operands of a predicate or function symbol, or logical connective; (3) change a predicate or function symbol, or logical connective, to one of the same type; (4) insert a logical negation; and (5) perform a local rewriting step to change a formula or a term to an equivalent one. Only (5) is semantics-preserving in general.

The goal of this experiment is to assess the impact of inserting theory lemmas dumped from a run of the solver on the original benchmark, on mutations of that benchmark. The rationale for inserting learned theory lemmas as possibly improving performance and/or predictability is that learned theory lemmas represent information the solver found useful when solving the original benchmark. The idea is that this information may also prove useful when solving a similar benchmark. It is also possible that the learned lemmas will mitigate the effect of the mutation.

Previous work by Whittemore et al. on reusing derived lemmas for SAT for a similar scenario (solving a sequence of related instances) requires tracking which lemmas are invalidated by changes in the formula [6]. In contrast, theory lemmas are, by definition, formulas which are valid in the solver's (possibly combined) background theory, without any other assumptions. Thus, it is always semantics-preserving to add a theory lemma, as it is always true modulo the theory, regardless of the rest of the benchmark formula. So no tracking is needed for theory lemmas (but would be needed for lemmas that follow from the input formula).

We modified the two open-source SMT solvers CVC3 and OPENSMT to dump learned theory lemmas, with the helpful advice of the authors of those tools. We then perform the following test for selected benchmarks (discussed below), using a timeout of 1 minute and a memory limit of 1GB:

1. Run the solver on the original benchmark. If this times out, abort the rest of the test.
2. Run the lemma-dumping modified version of the solver to generate theory lemmas (all other runs use the unmodified solver).
3. Insert theory lemmas into the benchmark as additional assumptions to obtain a modified benchmark. Run the solver on this modified benchmark.
4. Generate 11 mutants from the original benchmark, using the above mutations. For these experiments, we allowed 4 changes to each benchmark. For all divisions exception `QF_RDL`, we used 2 changes to formula structure, and 2 changes to term structure. For `QF_RDL`, changes to the term structure tend to take the benchmark out of the syntactic class for difference logic, so for `QF_RDL` we made 4 changes to the formula structure only.
5. Run the solver on each generated mutant.
6. Insert the lemmas dumped for the original benchmark into each mutant, and run the solver on each of the resulting benchmarks.

Dumping theory lemmas. As mentioned, we modified CVC3 and OPENSMT to dump learned theory lemmas, following helpful advice from the authors of those tools. Early experiments showed that inserting all learned theory lemmas into benchmarks tends to overwhelm the solver. So we just dump 10% of the learned theory lemmas. For CVC3, we dumped every 10th learned lemma. For OPENSMT, we dumped the 10% of learned theory lemmas with at most 2 literals which had the highest activity (as measured by `opensmt`'s internal measure of activity). Certainly one could be interested to compare alternative methods for selecting theory lemmas to dump. However, this is scheduled as future work. OPENSMT does not normally learn theory

name	orig	orig+lem	L	$\tilde{m}$	σ_m	$\tilde{l}$	σ_l	m/l	σ_m/σ_l
LamportBakery14	3.26	6.1	44	2.49	0.4	2.46	1.01		
LamportBakery15	2.19	2.22	58	1.97	0.29	1.93	0.27	1.02	1.05
OOO5	3.16	2.56	62	2.66	0.37	2.55	0.16	1.05	2.23
sorted_list_insert_noalloc3	4.86	4.52	61	4.13	0.37	4.34	0.36		1.03
vhard8	2.01	7.75	306	0.07	0.01	0.66	0.04		
OOO8	3.71	8.57	40	3.51	0.51	3.62	2.31		
ibm_cache_full_q_unbounded12	4.19	6.07	49	4.2	0.24	5.47	0.96		
sorted_list_insert_noalloc5	4.74	4.48	93	3.94	0.44	4.36	0.33		1.32
OOO6	3.22	6.43	40	2.78	0.41	3.99	1.44		
sorted_list_insert_noalloc6	7.03	4.42	239	3.79	1.4	4.26	0.44	1.15	3.15
vhard5	0.49	1.08	85	0.04	0.01	0.15	0.03		
ibm_cache_full_q_unbounded15	2.85	2.36	81	2.87	0.92	2.24	0.82	1.2	1.12
ibm_cache_full_q_unbounded14	4.04	4.16	35	4.05	0.75	4.13	0.36		2.06
vhard6	0.85	2.16	138	0.04	0.01	0.27	0.59		
ibm_cache_full_q_unbounded17	7.78	19.83	114	4.07	2.04	2.27	5.63		
ibm_cache_full_q_unbounded16	4.	4.04	35	4.01	0.74	4.04	0.17	1.06	4.3
vhard16	19.3	120.	2235	0.16	0.	7.43	0.01		
vhard9	2.77	15.39	427	0.08	0.01	0.95	0.03		
vhard18	29.23	120.	3151	0.19	0.	13.08	0.03		
vhard11	5.03	31.53	760	0.12	0.85	1.79	33.98(1)		

Figure 3: Results for CVC3: QF_UFIDL

lemmas outright (but rather, lemmas derived from theory lemmas by conflict analysis). We configured OPENSMT to learn theory lemmas of length at most 2, and kept that configuration for all runs of OPENSMT reported below.

Benchmark selection. The tests below were performed on a selection of benchmarks used in SMT-COMP 2009 (see, e.g., [1] and earlier papers for more on SMT-COMP). The selection process used was the following. We are looking at several mature example divisions: `QF_UFIDL`, `QF_AUFLIA`, `QF_LIA`, `QF_LRA`, and `QF_RDL`. CVC3 competed in all those divisions, while of these, `opensmt` competed just in `QF_LRA` and `QF_RDL`. For each solver and division in which it competed, we consider those benchmarks which it could solve in time between 1 second and 1 minute. A further issue we dealt with is that both SMT solvers sometimes introduce new symbols which make their way into theory lemmas. This is problematic, because the meanings and types of those symbols are not determined by the original benchmark. In the end, the approach we adopted was to try to translate away `ite`s (term-level if-then-else expressions) from benchmarks, since these seem to be the biggest (but not only) source of new symbols showing up in theory lemmas. CVC3 recently added a command-line option `+liftITE` that can be used to do this. In some cases, lifting `ite`s results in an explosion in the size of the formula, crashing the translating invocation of CVC3. In such cases, we excluded the benchmarks from our sample. The test machine for the experiments was a lightly-loaded Intel Core Dual CPU at 1.2GHz, with 1.5GB physical memory.

3.1 Empirical Results

Figures 3 through 10 summarize the results of these experiments (Figures 6 through 10 are relegated to the appendix for space reasons). Each figure corresponds to a single division and a single solver, except that for typographic reasons, the results for OPENSMT on `QF_RDL` are split over Figures 9 and 10. Each row in the table corresponds to a test on a single benchmark, as

name	orig	orig+lem	L	m̃	σ_m	l̃	σ_l	m/l	σ_m/σ_l
ParallelPrefixSum_live_blmc002	2.25	2.21	2	2.27	0.04	2.26	0.04		1.01
ParallelPrefixSum_safe_blmc008	9.57	9.49	6	9.55	0.59	9.62	0.56		1.04
FISCHER11-7-fair	99.89	15.67	180	92.08	47.59(1)	17.66	4.59		
FISCHER6-10-fair	69.24	41.13	293	56.03	29.68	52.89	23.86	1.32	1.24

Figure 4: Results for CVC3: QF_LIA

name	orig	orig+lem	L	m̃	σ_m	l̃	σ_l	m/l	σ_m/σ_l
storeinv_t1_pp_nf_ai_00009_001	47.6	16.19	11176	0.13	23.18	0.85	7.74	2.68	2.99
swap_t1_pp_nf_ai_00009_007	4.32	8.31	100	4.34	1.91	7.85	3.53		
pointer-safe-10	9.54	2.87	102	5.2	3.94	1.72	3.65	1.76	1.08
pointer-invalid-20	117.29	38.92	721	68.36	47.17(4)	31.46	32.88(1)		
pointer-invalid-10	10.72	1.17	126	4.65	4.7	1.73	4.1	1.97	1.14
pointer-safe-15	41.1	28.25	277	20.78	27.07	7.44	18.62	1.67	1.45
qlock-bug-5	19.48	5.7	312	17.86	37.84(1)	5.72	34.03(1)		
swap_t1_pp_nf_ai_00009_002	23.6	18.33	103	23.61	10.84	17.06	7.84	1.37	1.38
pointer-invalid-15	42.07	10.88	328	21.64	27.75	8.93	28.76	1.7	
qlock.base.5	14.68	12.45	215	12.77	33.94(1)	1.94	33.49(1)		
qlock-mutex-5	14.81	7.31	225	12.73	38.35(1)	1.85	33.6(1)		

Figure 5: Results for CVC3: QF_AUFLIA

described above. All times are given in seconds. The headings in the figure are as follows:

- `name`: the of the benchmark.
- `orig`: the time for the (unmodified) solver to solve the benchmark.
- `orig+lem`: the time for the solver to solve the modified version of the original benchmark, with theory lemmas inserted.
- `L`: the number of lemmas produced by the run of the lemma-generating version of the solver.
- m̃: the median of the times to solve the 11 generated mutants.
- σ_m: the standard deviation of those times.
- l̃: the median of the times to solve the 11 mutants with the same set of lemmas as above inserted into each mutant.
- σ_l: the standard deviation of those times.
- m/l: the ratio of total time to solve the 11 mutants to the total time to solve the 11 mutants with lemmas inserted. For readability, we only list this number (and similarly for the next column, for σ_m/σ_l) for rows where we have no missing data, and if it is greater than 1. This quantity represents the speedup using lemmas, and so we view this as a relative *performance* metric.
- σ_m/σ_l: the ratio of the above-defined standard deviations. We view this as a relative *predictability* metric.

Missing data. We lose data in this experiment for timeouts and memory outs, indicated in parentheses with the standard deviation; and occasionally where mutation takes a benchmark out of the syntactic class for the division, indicated in square brackets with the standard deviation. The latter problem just occurs for OPENSMT on `QF_LRA`, where the mutation occasionally creates divisions by zero.

Some observations about the data in the figures are warranted. First, it is not generally the case that either m/l or σ_m/σ_l is improved by inserting theory lemmas. The ratio σ_m/σ_l (computed only for tests with no censored data) is greater than 1 for 36% of the 68 total benchmarks (including tests with censored data). Similarly, the ratio m/l is greater than 1 for 35 %. We can observe, however, that in some cases, one or the other, or both, measures are improved. For example, in Figure 3, the `0005` benchmark shows a modest improvement in performance but a significant (> 2x) improvement in predictability. In some cases, such as `sorted_list_insert_noalloc5`, predictability improves even with an overall decline in performance. We can also see that inserting theory lemmas back into the original benchmark, while sometimes resulting in a significant slowdown (e.g., around 2x for `LamportBakery14` of Figure 3) can sometimes lead to big performance improvements: consider, for example, the results for `FISCHER11-7-fair` (Figure 4), where inserting theory lemmas leads to roughly a 6x speedup over the original benchmark; or for `pointer-invalid-15` (Figure 5), with around a 4x speedup.

So insertion of theory lemmas, while not generally helpful for performance or predictability, may have value as a heuristic for improving both. In our target use-model, a team making repeated calls to a solver can simply turn on "retain-lemmas" mode, and see if it improves performance or, over several runs, predictability. If not, the heuristic can be turned off. But if so, it may improve the end-user's experience for subsequent runs of the solver.

4 Conclusion and Future Work

We have considered preliminary data studying how various mutations to benchmarks affect the performance and reveal the predictability of SAT and SMT solvers, on collections of standard benchmarks. We have also considered one heuristic for improving predictability of SMT solvers, namely retaining learned theory lemmas from an original run to subsequent runs on similar benchmarks. This corresponds to a scenario where successive changes made by an end-user cause modest changes to the benchmark formula.

For future work, a more informed model for mutation is required, to ensure that the proposed methods apply in real-world situations. For such a model, it would be very useful to have or to be able to generate a sequence of formulas from successive small modifications to a verification model or other application-specific structure. This will enable more accurate further studies of methods to measure and improve solver predictability. One possibility might be to consider benchmarks from the same benchmark family (for both the SAT and SMT experiments), since these should exhibit some similarities. For the SMT experiments, it would be interesting to test whether or not adding theory lemmas learned from a run of the SMT solver on one benchmark can improve predictability for other benchmarks from the same family. Similarly, it would be interesting to do more thorough exploration of which lemmas to carry over from one run to the next, and whether lemmas learned by one solver can improve predictability for another.

Acknowledgments. Thanks to Clark Barrett and Roberto Bruttomesso for help modifying CVC3 and OPENSMT, respectively. Thanks also to Alberto Segre for consultation on statistics, and to Cesare Tinelli for discussion of the idea of considering predictability for SMT solvers.

References

[1] Clark Barrett, Morgan Deters, Albert Oliveras, and Aaron Stump. Design and results of the 3rd annual satisfiability modulo theories competition (SMT-COMP 2007). *International Journal on Artificial Intelligence Tools (IJAIT)*, 17(4):569–606, August 2008.

[2] Clark Barrett, Silvio Ranise, Aaron Stump, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2008.

[3] D. Berre and L. Simon. 55 Solvers in Vancouver: The SAT 2004 competition. In H. Hoos and D. Mitchell, editors, *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2004.

[4] E. Giunchiglia and M. Maratea. Planning as Satisfiability with Preferences. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence.*, pages 987–992. AAAI Press, 2007.

[5] S. Lahiri, S. Qadeer, and Z. Rakamaric. Static and Precise Detection of Concurrency Errors in Systems Code Using SMT Solvers. In A. Bouajjani and O. Maler, editors, *Computer Aided Verification, 21st International Conference, CAV 2009. Proceedings*, pages 509–524, 2009.

[6] J. Whittemore, J. Kim, and K. Sakallah. Satire: A new incremental satisfiability engine. In *Design Automation Conference, 2001. Proceedings*, pages 542 – 545, 2001.

[7] H. Zhang. Combinatorial designs by SAT solvers. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 17, pages 533–568. IOS Press, 2009.

A More Results from SMT Experiments

name	orig	orig+lem	L	$\tilde{m}$	σ_m	$\tilde{l}$	σ_l	m/l	σ_m/σ_l
sc-35.induction3	5.02	120.	94	1.45	2.72	1.48	57.06(4)		
sc-15.induction	1.49	0.94	16	0.32	0.54	0.33	0.36	1.24	1.5
uart-7.induction	2.08	1.78	78	0.67	0.67	0.64	0.66	1.05	
sc-12.induction	1.17	0.67	14	0.26	0.4	0.27	0.24	1.23	1.65
p2-zenonumeric_s6	2.79	7.77	27	1.78	0.4	3.23	1.54		
uart-6.induction	1.62	1.07	69	0.49	0.51	0.53	0.32	1.07	1.56
sc-14.induction	1.33	0.86	16	0.32	0.5	0.3	0.32	1.29	1.56
reint_to_least.base	7.06	7.2	30	0.21	0.79	0.23	34.31(1)		
pursuit-safety-7	5.38	1.03	50	1.65	8.9	0.58	0.46	6.22	19.28
sc-18.induction	1.91	1.18	19	0.4	0.76	0.41	0.46	1.31	1.64
sc-10.induction	0.79	0.54	12	0.22	0.26	0.24	0.19	1.14	1.35
uart-9.induction	2.98	3.32	100	0.92	1.04	1.09	1.27		
p-0-bucket_s7	5.3	4.6	58	5.33	0.04	4.74	0.39	1.1	
gasburner-prop3-16	0.75	0.77	20	0.53	19.63	0.63	12.71	1.46	1.54
gasburner-prop3-17	0.79	1.07	21	0.66	34.16(1)	0.72	36.55(1)		
clocksynchro_3clocks.main_invar.base	1.05	1.09	18	0.23	0.07	0.27	0.24		
p-driverlogNumeric_s7	61.48	120.	21	3.59	4.1	6.59	32.75(1)		
tgc_io-safe-13	11.41	6.6	70	2.03	2.41	2.83	3.26		
sc-24.induction	3.37	1.93	25	0.58	1.48	0.7	0.75	1.42	1.95
clocksynchro_9clocks.main_invar.base	7.78	7.72	51	0.78	0.37	0.94	2.14		
p4-zenonumeric_s5	59.2	90.93	68	11.49	25.67	12.33	32.11		
uart-8.base	17.43	15.73	279	1.61	3.54	4.44	3.57		
tgc_io-safe-18	42.	13.5	120	3.55	9.65	4.8	33.05(1)		
p6-zenonumeric_s5	65.28	120.	99	20.06	31.61(1)	22.37	43.77(3)		
sc-8.induction2	3.76	4.09	33	0.24	1.7	0.34	1.9		
simple_startup_3nodes.abstract.induct	22.71	80.12	431	0.24	0.09	0.47	0.38		
tgc_io-safe-20	68.34	5.74	144	4.85	15.38	4.96	33.37(1)		
simple_startup_4nodes.missing.induct	2.65	1.07	143	0.32	0.1	0.4	0.18		
uart-24.induction	18.58	60.41	435	5.16	6.02	5.22	18.92		
sc-19.induction	2.12	1.26	20	0.39	0.84	0.44	0.5	1.32	1.69
sc-21.induction3	12.72	30.18	111	0.86	32.94	0.96	14.68	1.94	2.24
uart-9.base	39.64	25.67	457	1.46	5.02	5.28	16.33		
synched.induction	3.36	3.84	49	0.27	2.46	0.29	34.17(1)		
sc-32.induction3	4.4	120.	82	1.29	2.26	1.32	51.89(3)		
uart-20.induction	13.01	29.44	318	3.56	4.18	3.74	10.54		
simple_startup_5nodes.missing.induct	7.36	3.29	333	0.45	0.17	0.61	0.47		
simple_startup_4nodes.abstract.induct	76.84	120.	1048	0.33	0.13	0.83	0.32		

Figure 6: Results for CVC3: QF_LRA

name	orig	orig+lem	L	$\tilde{m}$	σ_m	$\tilde{l}$	σ_l	m/l	σ_m/σ_l
fischer3-mutex-8	9.13	14.7	197	7.44	2.54	3.47	4.18	1.28	
orb02_700	4.32	4.23	32	4.71	0.16	4.49	0.23	1.04	
fischer6-mutex-6	7.16	14.68	169	8.17	4.3	11.2	4.85		
abz6_800	4.9	5.29	36	4.02	0.21	4.36	0.23		
orb07_330	5.63	6.41	42	4.7	0.42	5.36	0.42		
orb05_700	74.98	57.42	115	65.43	23.57(3)	68.67	15.15		
fischer9-mutex-5	9.12	15.05	126	21.87	18.73	19.	9.	1.45	2.08
fischer3-mutex-9	16.07	20.2	356	7.16	5.81	6.54	6.24	1.22	
fischer6-mutex-7	23.66	32.32	406	26.93	13.73	28.67	18.02		
fischer9-mutex-6	50.3	91.25	405	50.3	14.43	66.48	34.54(1)		
fischer3-mutex-10	23.39	50.63	572	19.75	9.75	5.11	16.36	1.29	
fischer6-mutex-8	103.89	120.	1209	42.34	35.89(1)	45.87	38.81(2)		
abz5_1400	27.59	120.	64	23.5	2.16	120.	0.(11)		
orb09_1100	59.45	120.	84	27.74	17.18	120.	0.(11)		
abz5_1000	12.53	21.89	59	39.31	10.57	29.32	6.32	1.2	1.67
fischer3-mutex-12	93.66	120.	1319	58.14	28.9	10.82	43.58(1)		
fischer3-mutex-13	118.06	120.	1734	100.14	44.77(3)	16.17	48.22(3)		
fischer3-mutex-11	40.47	119.12	902	17.9	19.61	9.07	35.27(1)		
orb04_850	25.93	42.75	53	19.98	3.02	42.59	10.21		
orb04_1200	29.05	120.	63	26.81	2.21	120.	0.(11)		
orb06_1200	46.71	120.	78	82.05	25.62	120.	4.77(10)		
orb05_1000	37.57	120.	62	60.41	13.01	120.	27.95(7)		
orb10_1100	37.18	120.	68	36.09	0.61	85.53	18.52(3)		

Figure 7: Results for CVC3: QF_RDL

name	orig	orig+lem	L	$\tilde{m}$	σ_m	$\tilde{l}$	σ_l	m/l	σ_m/σ_l
sc-10.induction	6.07	7.55	15	0.2	2.89	0.2	3.35		
pursuit-safety-10	1.27	1.36	18	0.54	1.33	0.59	1.06	1.1	1.25
p5-zenonumeric_s5	3.29	3.66	37	3.26	0.06	3.66	0.12		
pursuit-safety-11	1.42	1.65	17	0.66	2.87	0.74	1.72	1.35	1.66
uart-8.base	2.93	4.12	16	3.32	1.98	2.83	1.88	1.04	1.05
sc-11.induction	12.33	8.77	17	0.22	4.9	0.22	5.02		
p-0-bucket_s10	3.74	4.42	6	6.14	2.06	5.33	0.56	1.21	3.66
p-DepotsNum_s8.msat	3.54	2.57	32	1.86	0.45	1.69	0.45	1.07	
uart-7.induction	3.38	3.06	21	0.15	1.49	0.16	1.17	1.2	1.26
sc-12.induction	19.71	11.25	18	0.24	8.28	0.25	8.23	1.01	
uart-9.base	6.75	8.34	21	5.01	3.05	4.24	2.84	1.1	1.07
pursuit-safety-13	2.75	3.01	24	1.02	5.2	0.74	4.81	1.11	1.08
tgc_io-safe-18	5.2	3.13	36	3.06	1.34	3.	1.18		1.13
clocksynchro_9clocks.main_invar.base	5.07	6.4	6	0.41	3.15[1]	0.53	2.79[1]		
pursuit-safety-16	1.06	2.5	16	1.62	12.63	1.75	20.28		
tgc_io-safe-20	5.7	7.15	39	4.88	1.88	4.36	2.38		
p-0-bucket_s13	8.	6.24	6	12.61	3.13	10.93	5.03	1.07	
p7-driverlogNumeric_s7	5.23	12.96	45	2.19	1.51	2.39	2.27		
sc-15.induction	15.12	42.9	18	0.3	17.62	0.31	18.03		
uart-9.induction	7.72	9.21	26	0.22	3.3	0.21	2.91	1.13	1.13
simple_startup_3nodes.abstract.induct	8.81	9.02	24	0.21	0.6[1]	0.19	0.52[1]		
sc-14.induction	39.51	30.79	20	0.28	11.12	0.28	13.86		
sc-12.induction3	11.55	13.79	13	0.31	6.84	0.3	4.61	1.43	1.48
simple_startup_4nodes.missing.induct	12.17	15.59	29	0.28	3.11[1]	0.29	2.45[1]		
sc-10.base	14.21	13.08	15	7.03	3.95	7.96	4.67		
pursuit-safety-15	1.67	2.25	17	1.58	9.62	1.57	11.97		
sc-14.induction3	13.74	22.94	14	0.35	12.87	0.35	10.9	1.12	1.18
sc-18.induction	110.77	116.39	25	0.38	44.05	0.37	35.59	1.22	1.23
p-2-bucket_s11	26.33	24.54	8	25.81	1.52	25.28	2.01		
simple_startup_5nodes.missing.induct	55.16	46.07	43	0.38	10.74[1]	0.39	11.17[1]		
sc-12.base	32.94	33.58	19	14.3	10.19	18.71	10.88		
sc-19.induction	90.19	114.56	24	0.39	50.32(1)	0.38	49.99(1)		
p7-driverlogNumeric_s8	13.03	30.67	61	2.42	4.97	2.53	15.52		
sc-17.induction2	70.35	54.6	24	0.38	27.86	0.38	28.62	1.02	
sc-19.induction2	60.03	120.	22	0.48	45.44	0.43	47.49		
simple_startup_4nodes.abstract.induct	50.11	44.52	38	0.28	2.83[1]	0.28	3.38[1]		
opt1217–6	45.69	42.71	313	7.43	19.52	5.15	19.72	1.02	
uart-13.base	47.2	53.14	44	17.96	12.91	17.7	10.83	1.08	1.19
opt1217–11	57.04	53.3	332	7.48	21.33	6.79	20.62	1.05	1.03
sc-14.base	74.98	58.92	21	29.43	20.23	36.85	19.05		1.06
sc-15.base	94.18	95.45	22	45.51	22.94	34.34	21.29	1.14	1.07

Figure 8: Results for opensmt: QF_LRA

name	orig	orig+lem	L	$\tilde{\mathrm{m}}$	σ_m	$\tilde{\mathrm{l}}$	σ_l	m/l	σ_m/σ_l
orb07_430	0.56	0.93	6	0.34	0.31	0.35	0.42		
orb08_930	4.63	3.91	6	1.56	1.53	1.66	5.		
fischer3-mutex-12	3.79	3.32	21	0.83	0.89	1.04	0.83	1.01	1.06
fischer3-mutex-15	6.36	6.51	27	1.99	1.5	1.16	1.93		
orb01_1200	0.88	0.3	6	0.35	0.23	0.31	0.05	1.42	4.51
skdmxa-3x3-6.base	8.02	8.07	5	8.31	2.24	8.27	2.24		
fischer3-mutex-14	4.49	4.02	23	1.37	1.39	1.57	1.2		1.15
abz6_943	3.29	3.76	6	0.38	1.17	0.4	1.3		
orb10_900	2.5	3.02	6	8.93	3.05	9.81	3.25		
fischer3-mutex-13	4.73	5.03	23	1.24	1.36	0.9	0.98	1.24	1.38
fischer9-mutex-8	5.4	4.71	30	1.72	1.91	1.6	1.73		1.1
skdmxa-3x3-7.base	9.42	9.34	5	10.26	2.81	10.48	2.82		
fischer6-mutex-9	6.6	4.27	23	1.47	1.62	1.62	1.15		1.4
abz7_500	5.76	5.83	6	5.7	0.95	5.79	1.22		
orb09_900	6.77	6.08	7	3.36	1.7	4.2	2.55		
orb10_1000	3.62	3.09	6	1.03	2.21	1.1	1.11	1.17	1.98
skdmxa-3x3-8.base	12.29	11.94	5	12.95	3.8	12.85	3.75		1.01
orb06_1100	2.02	3.7	6	0.81	0.85	0.54	1.14		
fischer3-mutex-16	8.16	9.42	31	2.08	2.32	1.7	2.73		
orb04_1005	27.42	6.25	9	5.26	5.07	5.47	6.22[1]		
skdmxa-3x3-9.base	14.87	14.87	5	16.14	4.73	16.08	4.69	1.01	
fischer3-mutex-17	10.14	7.96	29	2.21	3.02	2.36	2.34	1.11	1.28
orb10_944	14.39	14.64	6	2.33	4.67	2.45	6.32		
orb02_888	3.56	3.33	6	0.37	1.94	0.36	2.78		
fischer9-mutex-9	10.56	18.32	36	3.22	3.92	3.89	3.32		1.18
fischer3-mutex-18	13.05	13.37	36	2.89	4.11	2.27	3.82	1.01	1.07
skdmxa-3x3-5.induction	28.12	17.71	7	19.72	7.86	17.12	7.06	1.14	1.11
skdmxa-3x3-10.base	19.6	19.4	5	19.71	6.05	19.96	6.05		
fischer6-mutex-10	7.65	8.72	29	2.76	2.4	2.4	2.81		

Figure 9: Results for opensmt: QF_RDL

name	orig	orig+lem	L	$\tilde{m}$	σ_m	$\tilde{l}$	σ_l	m/l	σ_m/σ_l
fischer9-mutex-10	22.02	22.35	45	6.04	9.93	7.94	4.99	1.4	1.98
orb03_1100	10.57	0.38	8	0.4	1.21	0.41	0.59	1.83	2.03
orb07_397	10.62	17.67	6	0.66	5.9	0.78	7.12		
abz5_1200	8.91	9.01	7	0.47	4.2	0.46	4.09	1.02	1.02
orb03_950	14.55	13.53	7	11.96	1.95	11.74	3.18		
fischer3-mutex-19	18.64	16.42	43	2.6	5.59	3.74	5.08		1.09
skdmxa-3x3-11.base	23.17	23.28	5	25.02	8.09	25.05	8.		1.01
orb09_934	30.93	10.48	9	0.52	3.99	0.48	9.34		
abz7_800	19.33	4.34	6	3.63	6.19	4.24	0.8	1.61	7.69
orb08_888	17.67	19.	8	4.52	7.19	4.66	6.74	1.07	1.06
skdmxa-3x3-12.base	30.	30.1	5	31.34	10.19	30.93	10.08	1.01	1.01
fischer6-mutex-11	12.54	26.58	35	3.3	4.36	3.69	9.3		
skdmxa-3x3-13.base	34.69	35.08	5	35.56	12.31	35.69	12.34		
fischer3-mutex-20	19.34	24.25	42	2.03	6.02	3.46	5.94		1.01
orb01_1100	21.69	18.04	8	1.19	2.23	0.88	8.21		
fischer6-mutex-12	43.47	25.85	52	6.04	12.27	5.03	9.14	1.14	1.34
orb05_887	7.7	17.2	8	0.42	7.3	0.36	5.14	1.37	1.42
skdmxa-3x3-14.base	43.51	43.06	5	45.87	15.52	45.92	15.62		
skdmxa-3x3-5	32.3	40.95	39	27.94	19.21	34.88	26.52		
fischer6-mutex-13	30.86	52.75	54	8.69	14.24	6.94	19.28		
abz5_1234	30.29	17.22	9	0.38	16.33	0.4	5.76	2.68	2.83
skdmxa-3x3-6.induction	48.16	33.19	9	35.92	15.68	30.94	13.51	1.01	1.16
skdmxa-3x3-15.base	56.2	55.89	5	57.54	20.43	57.21	20.48		
skdmxa-3x3-16.base	75.21	75.66	5	73.81	26.43	73.93	26.5		
fischer6-mutex-16	108.07	120.	73	34.07	40.1	42.84	41.86(1)		
fischer9-mutex-11	54.97	43.05	65	16.96	15.97	9.5	19.75		
fischer6-mutex-15	83.54	78.13	70	20.37	27.43	35.25	28.53		
fischer6-mutex-14	106.9	54.42	67	17.41	17.79	14.53	19.82		

Figure 10: Results for opensmt: QF_RDL