Exploring Predictability of SAT/SMT solvers

Robert Brummayer[1] Duckki Oe[2] Aaron Stump[2]

[1]Johannes Kepler University
Linz, Austria

[2]The University of Iowa
Iowa City, Iowa, USA

EMSQMS'10

Speed is Everything!

- SAT/SMT solvers have made tremendous advances in performance.
- $>$ 100k variables, $>$ 1M clauses.
- Due to:
  - algorithmic advances (clause learning, integration, theory solvers, etc.)
  - good heuristics
  - good engineering
- SAT Competition, SMT-COMP reward speed.
- Applications like program verification need raw power.

Is Speed Everything?

- State-of-the-art on SMT-COMP stagnant.
- But new theories: **expressiveness**.
- For some applications, many easy queries: **embeddability**.
- Even for tough benchmarks, **predictability** an issue.
  - Steve Miller (Rockwell Collins): solver performance is unpredictable.
  - Small change to model => big change to run-time.
  - Problematic for development.

This Talk: Exploring Predictability

Motivation: incrementally changing formulas.

- planning/AI applications
  - answer queries based on policies, observations.
  - incrementally changing observations => similar queries.
- software verification
  - call solver to compare code to spec.
  - gradually evolving code/spec => evolving formulas.
- static analysis
  - analyze paths through code.
  - changing code => gradually changing queries.

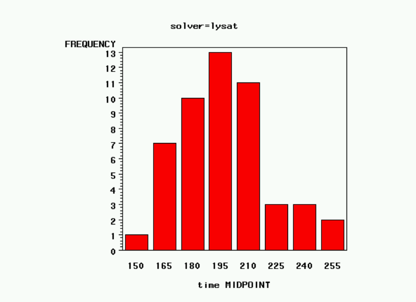
Issues with unpredictability.

- less predictable => harder to embed (e.g., in a compiler).
- end-user frustration.

Measuring Predictability

Population: set of similar instances

- Pick a SAT formula as *seed formula*.
- Generate 50 random variations.
- Run solver to get distribution of solving times.
- Measure of predictability: the standard deviation.



Types of changes

Semantics-preserving:

- *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

Semantics-modifying:

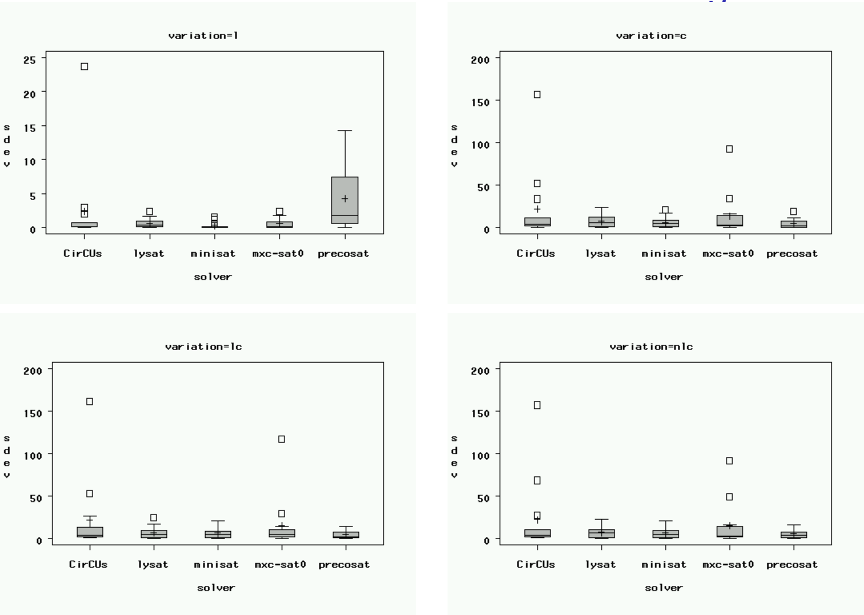
- *nlcx* : *nlc* + one literal of clause is changed (0.01%)
- *nlca* : *nlc* + one literal is dropped/added to clause (0.01%)

- unary clauses are not modified
- preserves literal/clause ratio

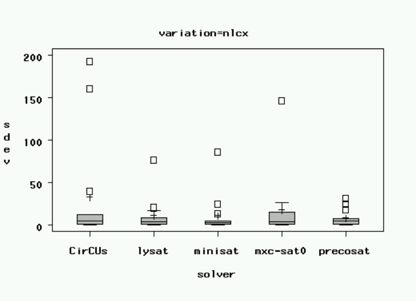
Experiments

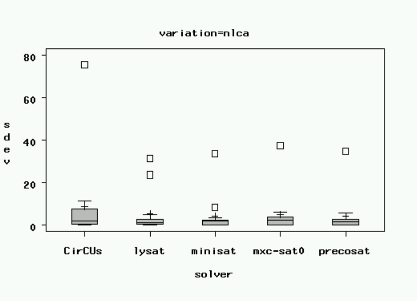
- 5 solvers: high ranking in SAT Competition 2009.
- 13 seed formulas: 5 easy, 6 medium, 2 hard.
- Generate 50 instances for each change-type.
- For each seed formula, each solver:
  - Run solver on the 50 instances for the seed.
  - Compute std. dev. of runtimes.
- Graph all std. devs.

Run-Time Std Devs – Semantics Preserving

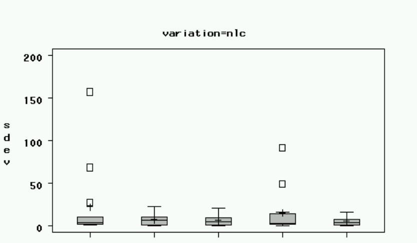


Run-Time Std Devs – Semantics Modifying



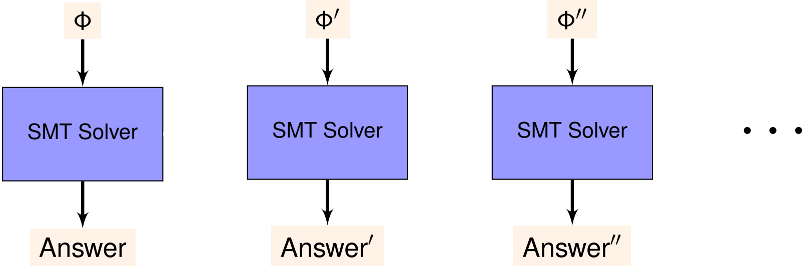


Compare with:



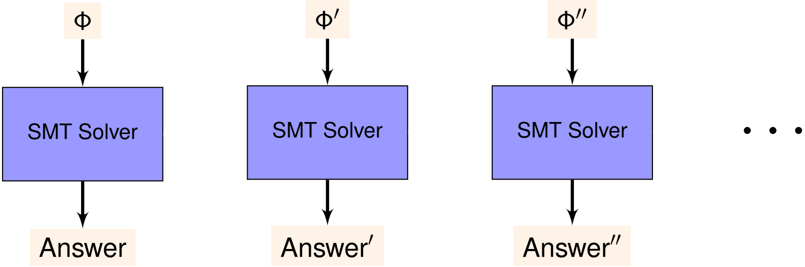
Improving Predictability for SMT

Multiple runs on similar formulas:



Improving Predictability for SMT

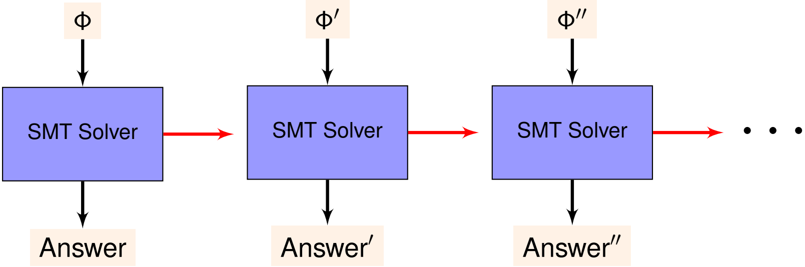
Multiple runs on similar formulas:



Idea: pass along some theory lemmas.

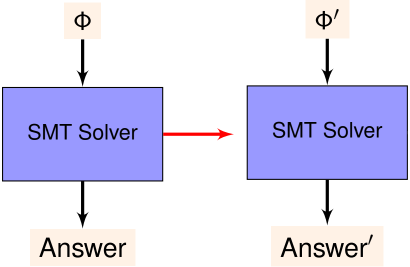
Improving Predictability for SMT

Multiple runs on similar formulas:



Idea: pass along some theory lemmas.

Dumping theory lemmas



- theory lemmas valid => always safe to add.
- helpful once => may be helpful again.
- lots of lemmas => just dump 10%.

Experiments

- Modified CVC3 and `opensmt` to dump lemmas.
  - Open-source tools.
  - Very helpful developers (thanks Clark Barrett, Roberto Bruttomesso).
  - Not too hard to modify.
- Selected seed formulas from example divisions.
- Generate 11 mutants for each seed.
  - mutator based on Robert's SMT fuzzer/delta-debugger.
  - small number (4) of semantics-modifying changes.
- For each seed formula:
  1. Run solver on seed formula.
  2. Re-run on seed, dumping lemmas.
  3. Re-run on seed + lemmas.
  4. Run mutants.
  5. Re-run mutants + lemmas (from seed).
- Compare times for mutants, mutants + lemmas.

Results for CVC3: QF_UFIDL

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_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		
cache_unbounded12	4.19	6.07	49	4.2	0.24	5.47	0.96		
sorted_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_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		
cache_unbounded15	2.85	2.36	81	2.87	0.92	2.24	0.82	1.2	1.12
cache_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		
cache_unbounded17	7.78	19.83	114	4.07	2.04	2.27	5.63		
cache_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)		

orig = solver on seed

L = num dumped lemmas

$\tilde{m}$ = mean time, mutants

$\tilde{l}$ = mean time, mutants+lemmas

σ_m = std. dev, mutants

σ_l = std. dev, mutants+lemmas

Results for `opensmt`: QF_LRA

name	orig	orig+lem	L	$\tilde{m}$	σ_m	$\tilde{l}$	σ_l	m/l	σ_m/σ_l
sc-10.ind	6.07	7.55	15	0.2	2.89	0.2	3.35		
safety-10	1.27	1.36	18	0.54	1.33	0.59	1.06	1.1	1.25
p5-zenonum_s5	3.29	3.66	37	3.26	0.06	3.66	0.12		
safety-11	1.42	1.65	17	0.66	2.87	0.74	1.72	1.35	1.66
uart-8.b	2.93	4.12	16	3.32	1.98	2.83	1.88	1.04	1.05
sc-11.ind	12.33	8.77	17	0.22	4.9	0.22	5.02		
p-0_s10	3.74	4.42	6	6.14	2.06	5.33	0.56	1.21	3.66
Dep_s8.ms	3.54	2.57	32	1.86	0.45	1.69	0.45	1.07	
uart-7.ind	3.38	3.06	21	0.15	1.49	0.16	1.17	1.2	1.26
sc-12.ind	19.71	11.25	18	0.24	8.28	0.25	8.23	1.01	
uart-9.b	6.75	8.34	21	5.01	3.05	4.24	2.84	1.1	1.07
safety-13	2.75	3.01	24	1.02	5.2	0.74	4.81	1.11	1.08
io-safe-18	5.2	3.13	36	3.06	1.34	3.	1.18		1.13
9clks.inv.b	5.07	6.4	6	0.41	3.15[1]	0.53	2.79[1]		
safety-16	1.06	2.5	16	1.62	12.63	1.75	20.28		
io-safe-20	5.7	7.15	39	4.88	1.88	4.36	2.38		
p-0_s13	8.	6.24	6	12.61	3.13	10.93	5.03	1.07	
p7-drv_s7	5.23	12.96	45	2.19	1.51	2.39	2.27		
sc-15.ind	15.12	42.9	18	0.3	17.62	0.31	18.03		
uart-9.ind	7.72	9.21	26	0.22	3.3	0.21	2.91	1.13	1.13
3nodes.ind	8.81	9.02	24	0.21	0.6[1]	0.19	0.52[1]		
. . .									

orig	=	solver on seed	L	=	num dumped lemmas
$\tilde{m}$	=	mean time, mutants	$\tilde{l}$	=	mean time, mutants+lemmas
σ_m	=	std. dev, mutants	σ_l	=	std. dev, mutants+lemmas

Conclusion

- Speed is not everything.
- Attributes like predictability also important.
- Experiments: SAT solvers differ in predictability.
- Passing theory lemmas can help SMT:
  - can improve performance a little (15-35%).
  - can improve predictability (3x, 3.5x).
  - but not predictably(!).
- Future work: try to improve predictability.
  - trade some performance for predictability.
  - canonical forms for SAT formulas?
  - run seed, mutant formula together?
  - use formula diffs? proofs?

Conclusion

- Speed is not everything.
- Attributes like predictability also important.
- Experiments: SAT solvers differ in predictability.
- Passing theory lemmas can help SMT:
  - can improve performance a little (15-35%).
  - can improve predictability (3x, 3.5x).
  - but not predictably(!).
- Future work: try to improve predictability.
  - trade some performance for predictability.
  - canonical forms for SAT formulas?
  - run seed, mutant formula together?
  - use formula diffs? proofs?

Maybe you want to try to improve predictability!