

Benchmarking Solvers, SAT-style

{Martin Nyx Brain, James H. Davenport & Alberto Griggio}¹
University of Oxford, University of Bath, Fond. Bruno Kessler
Martin.Brain@cs.ox.ac.uk,
J.H.Davenport@bath.ac.uk,
Griggio@FBK.eu

29 July 2017

¹Thanks to EU H2020-FETOPEN-2016-2017-CSA project SC^2 (712689) and the many partners on that project: www.sc-square.org

(Caricature of) Attitudes

Guess which is which?

(Caricature of) Attitudes

Guess which is which?

SC I want to win the next competition, which will have a mixture of hard and easy problems, and be judged on time-to-solve

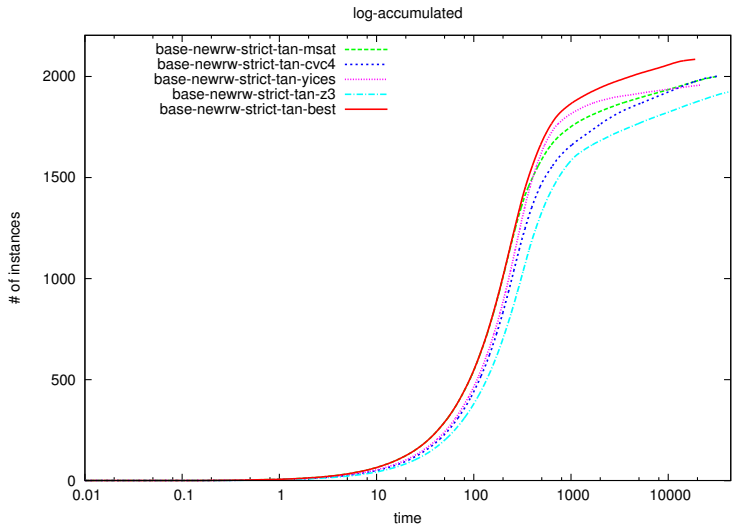
(Caricature of) Attitudes

Guess which is which?

- SC I want to win the next competition, which will have a mixture of hard and easy problems, and be judged on time-to-solve
- SC I want to submit a paper with timings that make my algorithm look good (on hard problems, ideally ones other people can't solve)

The SAT community, and hence the SMT community, have substantial experience in benchmarking solvers against each other on large sample sets, and publishing summaries, whereas the computer algebra community tends to time solvers on a small set of problems, and publishing individual times, with, at best, selective comparison.

Survivor plot



- 1 For each method separately

- 1 For each method separately
 - 1 Solve each problem p_i , noting the time t_i (up to some threshold T).

- ① For each method separately
 - ① Solve each problem p_i , noting the time t_i (up to some threshold T).
 - ② Sort the t_i into increasing order (discarding the time-out ones).

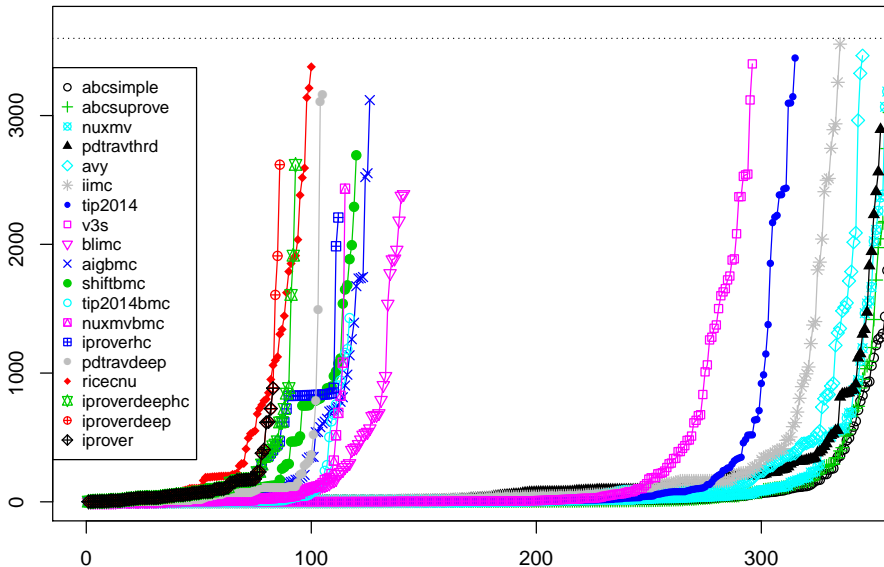
- 1 For each method separately
 - 1 Solve each problem p_i , noting the time t_i (up to some threshold T).
 - 2 Sort the t_i into increasing order (discarding the time-out ones).
 - 3 Plot the points $(t_1, 1)$, $(t_1 + t_2, 2)$ etc., and in general $(\sum_{i=1}^k t_i, k)$.

- 1 For each method separately
 - 1 Solve each problem p_i , noting the time t_i (up to some threshold T).
 - 2 Sort the t_i into increasing order (discarding the time-out ones).
 - 3 Plot the points $(t_1, 1)$, $(t_1 + t_2, 2)$ etc., and in general $(\sum_{i=1}^k t_i, k)$.
- 2 Place all the plots on the same axes, optionally (as we did) using a logarithmic scale for time.

- 1 For each method separately
 - 1 Solve each problem p_i , noting the time t_i (up to some threshold T).
 - 2 Sort the t_i into increasing order (discarding the time-out ones).
 - 3 Plot the points $(t_1, 1)$, $(t_1 + t_2, 2)$ etc., and in general $(\sum_{i=1}^k t_i, k)$.
- 2 Place all the plots on the same axes, optionally (as we did) using a logarithmic scale for time.

N.B. There is therefore no guarantee that the same problems were used to produce time results from different solvers.

HWMCC'15 Cactus SINGLE Track SAT+UNSAT



- 1 For each method separately

- 1 For each method separately
 - 1 Solve each problem p_i , noting the time t_i (up to some threshold T).

- 1 For each method separately
 - 1 Solve each problem p_i , noting the time t_i (up to some threshold T).
 - 2 Sort the t_i into increasing order (discarding the time-out ones).

- 1 For each method separately
 - 1 Solve each problem p_i , noting the time t_i (up to some threshold T).
 - 2 Sort the t_i into increasing order (discarding the time-out ones).
 - 3 Plot the points $(1, t_1,)$, $(2, t_1 + t_2, 2)$ etc., and in general $(k, \sum_{i=1}^k t_i)$.

- 1 For each method separately
 - 1 Solve each problem p_i , noting the time t_i (up to some threshold T).
 - 2 Sort the t_i into increasing order (discarding the time-out ones).
 - 3 Plot the points $(1, t_1)$, $(2, t_1 + t_2)$ etc., and in general $(k, \sum_{i=1}^k t_i)$.
- Or Plot the points $(1, t_1)$, $(2, t_2)$ etc., and in general (k, t_k) .

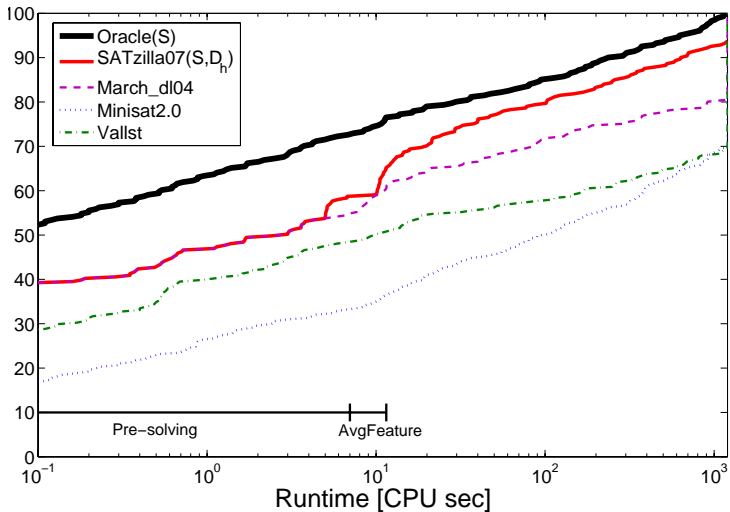
- 1 For each method separately
 - 1 Solve each problem p_i , noting the time t_i (up to some threshold T).
 - 2 Sort the t_i into increasing order (discarding the time-out ones).
 - 3 Plot the points $(1, t_1)$, $(2, t_1 + t_2)$ etc., and in general $(k, \sum_{i=1}^k t_i)$.
Or Plot the points $(1, t_1)$, $(2, t_2)$ etc., and in general (k, t_k) .
- 2 Place all the plots on the same axes

- 1 For each method separately
 - 1 Solve each problem p_i , noting the time t_i (up to some threshold T).
 - 2 Sort the t_i into increasing order (discarding the time-out ones).
 - 3 Plot the points $(1, t_1)$, $(2, t_1 + t_2)$ etc., and in general $(k, \sum_{i=1}^k t_i)$.
Or Plot the points $(1, t_1)$, $(2, t_2)$ etc., and in general (k, t_k) .
- 2 Place all the plots on the same axes
 - * Again, logarithmic time is possible.

- 1 For each method separately
 - 1 Solve each problem p_i , noting the time t_i (up to some threshold T).
 - 2 Sort the t_i into increasing order (discarding the time-out ones).
 - 3 Plot the points $(1, t_1)$, $(2, t_1 + t_2)$ etc., and in general $(k, \sum_{i=1}^k t_i)$.
Or Plot the points $(1, t_1)$, $(2, t_2)$ etc., and in general (k, t_k) .
- 2 Place all the plots on the same axes
 - * Again, logarithmic time is possible.

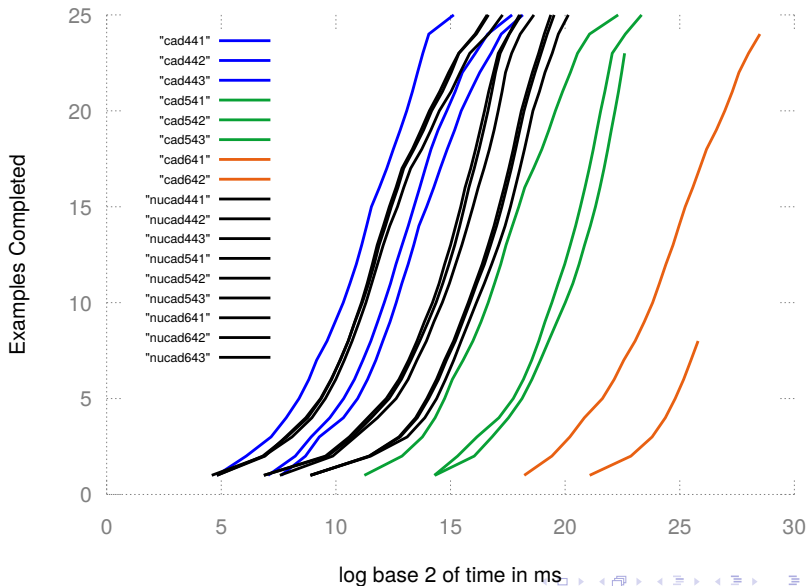
N.B. There is therefore no guarantee that the same problems were used to produce time results from different solvers.

Cumulative Density [XHHLB08]



Example from ISSAC (Brown)

Number of Examples Completed as a Function of Time



Virtual Best Solver/“Oracle”

Virtual Best Solver/“Oracle”

The SAT competition has taken to including a “virtual best solver” (VBS) which is synthesised from the other results by taking the minimum (across all solvers tested) time taken to solve every given benchmark. Thus the VBS time is always equal to the time of some real solver, but which one will change by the benchmark (measuring how often each solver is the VBS is also an interesting metric). The VBS can be added to the survivor/cactus plot, or indeed CDF, to get a feeling for the variability between solvers.

Virtual Best Solver/“Oracle”

The SAT competition has taken to including a “virtual best solver” (VBS) which is synthesised from the other results by taking the minimum (across all solvers tested) time taken to solve every given benchmark. Thus the VBS time is always equal to the time of some real solver, but which one will change by the benchmark (measuring how often each solver is the VBS is also an interesting metric). The VBS can be added to the survivor/cactus plot, or indeed CDF, to get a feeling for the variability between solvers. We often count how often a solver is the VBS.

Virtual Best Solver/“Oracle”

The SAT competition has taken to including a “virtual best solver” (VBS) which is synthesised from the other results by taking the minimum (across all solvers tested) time taken to solve every given benchmark. Thus the VBS time is always equal to the time of some real solver, but which one will change by the benchmark (measuring how often each solver is the VBS is also an interesting metric). The VBS can be added to the survivor/cactus plot, or indeed CDF, to get a feeling for the variability between solvers. We often count how often a solver is the VBS. A variation on counting is provided by [JLMS16], who measure how often a solver is within one second of being VBS. Their justification is “The constant of one second was chosen since we consider a smaller difference as insignificant, especially in the context of 800 second time-out”.

Multiple Copies

One of the effects of having a solution process whose running time is widely variable is that one may well not be best served by just running the process to termination.

Multiple Copies

One of the effects of having a solution process whose running time is widely variable is that one may well not be best served by just running the process to termination. In the case of a single processor, this issue was considered by [LSZ93], who suggested (and indeed proved almost-optimality) running the process up to certain time limits and then starting afresh, where the limits were of the form

$T, T, 2T, T, T, 2T, 4T, T, T, 2T, T, T, 2T, 4T, 8T, \dots$, where T is some arbitrary unit.

Multiple Copies

One of the effects of having a solution process whose running time is widely variable is that one may well not be best served by just running the process to termination. In the case of a single processor, this issue was considered by [LSZ93], who suggested (and indeed proved almost-optimality) running the process up to certain time limits and then starting afresh, where the limits were of the form

$T, T, 2T, T, T, 2T, 4T, T, T, 2T, T, T, 2T, 4T, 8T, \dots$, where T is some arbitrary unit.

This is in fact the default behaviour in MiniSAT 2.2.0, where it is known as Luby (though T is in fact measured in terms of conflicts rather than time, and it's not a complete restart that is performed, as certain learned clauses are kept).

Parallel running

These days, with processors getting more numerous rather than faster, we might consider running multiple copies in parallel. To see how this might help, consider the trivial case of a process whose running time is $1, K, K^2$ with equal probability. Then the average time to solution is $\frac{1}{3}(1 + K + K^2) = 37$ when $K = 10$.

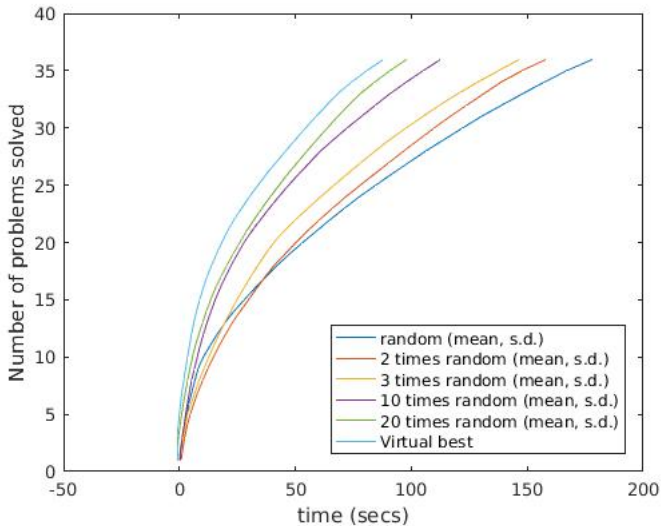
Parallel running

These days, with processors getting more numerous rather than faster, we might consider running multiple copies in parallel. To see how this might help, consider the trivial case of a process whose running time is $1, K, K^2$ with equal probability. Then the average time to solution is $\frac{1}{3}(1 + K + K^2) = 37$ when $K = 10$. Running two copies and aborting the other when one finds the solution has an average time to solution of $\frac{1}{9}(5 + 3K + K^2) = 15$ when $K = 10$, so the CPU cost is 30 units, still less than the sequential cost. Similarly, three copies gives $\frac{1}{27}(19 + 7K + K^2) = 7$ when $K = 10$, so the CPU cost is 21 units, even better. For $K = 10$, the minimum is achieved at 8-fold parallelism, with time-to-solution 1.36 units, and a CPU cost of 10.9 units.

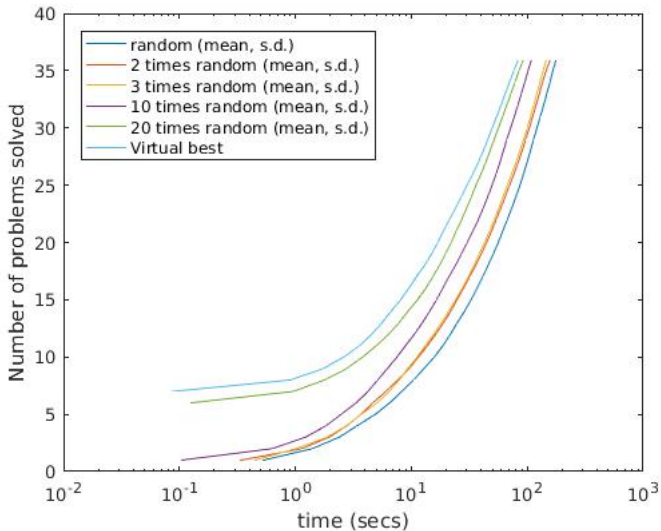
Parallel running

These days, with processors getting more numerous rather than faster, we might consider running multiple copies in parallel. To see how this might help, consider the trivial case of a process whose running time is $1, K, K^2$ with equal probability. Then the average time to solution is $\frac{1}{3}(1 + K + K^2) = 37$ when $K = 10$. Running two copies and aborting the other when one finds the solution has an average time to solution of $\frac{1}{9}(5 + 3K + K^2) = 15$ when $K = 10$, so the CPU cost is 30 units, still less than the sequential cost. Similarly, three copies gives $\frac{1}{27}(19 + 7K + K^2) = 7$ when $K = 10$, so the CPU cost is 21 units, even better. For $K = 10$, the minimum is achieved at 8-fold parallelism, with time-to-solution 1.36 units, and a CPU cost of 10.9 units. The break even point for two-fold parallel running is $K = \frac{1}{2}(1 + \sqrt{37}) \approx 4.5$, and three-fold running is $K = 4$. It is worth noting, though, that a single Luby process with $T = \frac{1}{3}$ (to avoid $T = 1$ getting lucky) achieves an average time to solution (and cost) of ≈ 9 .

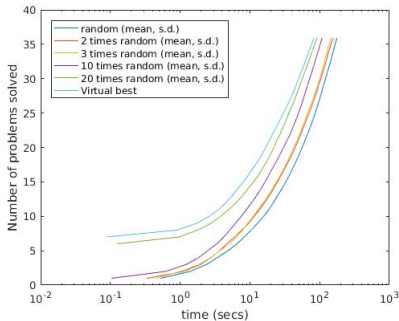
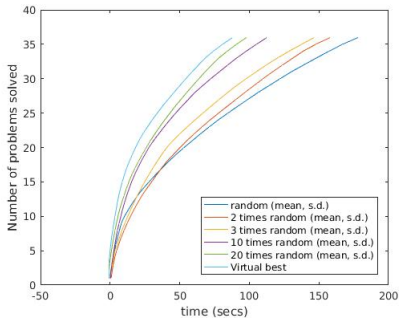
Normal Distributions: plot



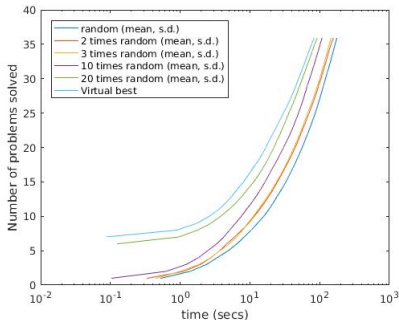
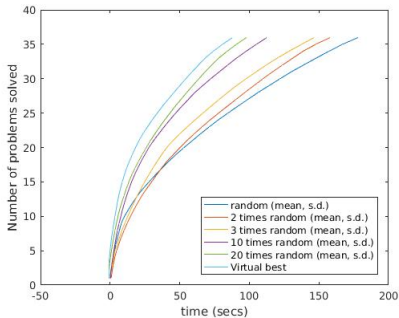
Normal Distributions: log time plot



Normal Distributions: compared

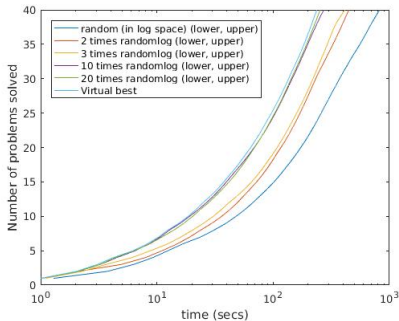
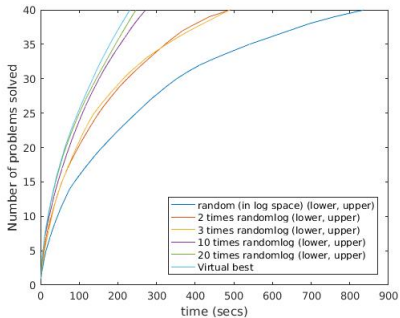


Normal Distributions: compared

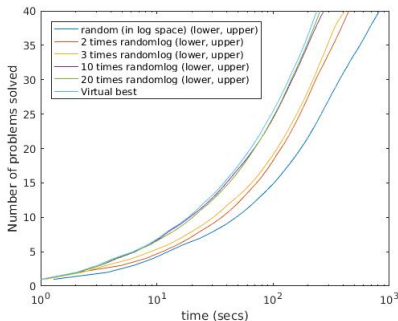
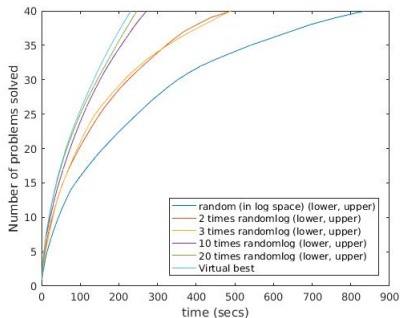


Note that we get very different conclusions from the two.

Uniform in $\log(t)$



Uniform in $\log(t)$



Seems that running twice and running thrice were very similar, and in fact that running twice was almost half the time of running once, thus meaning that they were almost equivalent in cost.

Uniform in $\log(t)$: Analysis



In fact, this model is susceptible to algebraic treatment, and the formulae (running from 1 to B seconds, with numeric values for $B = 10$) are as follows:


$$\begin{aligned} \text{once} &= \frac{B-1}{\log B} \approx 3.9087 \\ \text{twice} &= \frac{2}{(\log B)^2} (B - (\log B + 1)) \approx 2.5264 \\ \text{thrice} &= \frac{6}{(\log B)^3} (B - (\frac{1}{2} \log B + \log B + 1)) \approx 1.9887 \end{aligned}$$

Hence in fact the “running thrice” number is approximately correct, at one-half the elapsed time of running once.

- 1 Brown was using (similar) random examples in each line
- 2 [BH15] had over 300 examples, and this is not uncommon:
 - * Indeed a talk today had > 5000 examples.
- 3 My slide had a misleading conclusion from only 20 samples.
- 4 Plotting time and $\log(\text{time})$ gives very different graphs

Questions?

-  A. Biere and K. Heljanko.
Hardware Model Checking Competition Report HWMCC'15.
<http://fmv.jku.at/hwmcc15/Biere-HWMCC15-talk.pdf>,
2015.
-  M. Janota, I. Lynce, and J. Marques-Silva.
Algorithms for computing backbones of propositional formulae.

AI Communications, 28:161–177, 2016.
-  M. Luby, A. Sinclair, and D. Zuckerman.
Optimal Speedup of Las Vegas algorithms.
Information Processing Letters, 47:173–180, 1993.



L. Xu, F. Hutter, H.H. Hoos, and K. Leyton-Brown.
SATzilla: Portfolio-based Algorithm Selection for SAT.
Journal of Artificial Intelligence Research, 32:565–606, 2008.