Benchmarking

MJB/JHD/AG

June 1, 2017

Abstract

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.

This paper aims to document the SAT community practice for the benefit of the computer algebra community.

Implicitly underlying benchmarking for solvers is the following hypothesis.

Hypothesis 1 *Our solvers will be faced in practice (read "the next competition") with problems whose time-to-solve distribution is the same as that for the benchmark set.*

A necessary condition is that one has a large benchmark set, ideally in the thousands. Sampling uniformly is always a question of judgement.

1 "Cactus" or "Survivor" plots

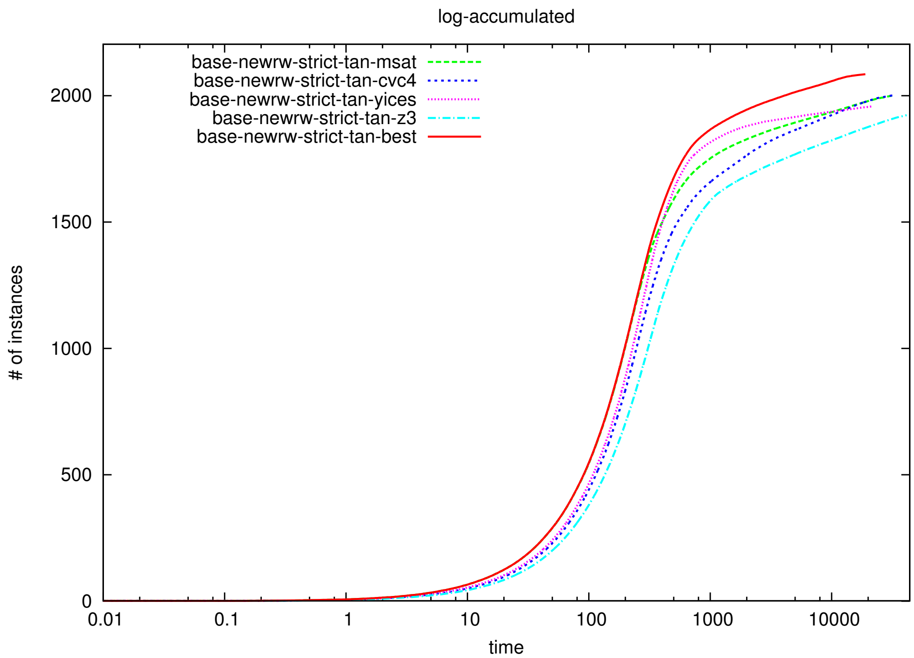
Figure 1 is a typical survivor[1] plot produced in the SMT community. The methodology for producing these, given a large benchmark set of problems, is as follows.

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

[1] "Cactus" plots are the same with the axes flipped. Some cactus plots can be seen at `http://fmv.jku.at/hwmcc15/Biere-HWMCC15-talk.pdf`.

Figure 1: A Survivor Plot



From 1 we can deduce that, up to 100 seconds, the solvers are pretty similar, and with a time budget of 100 seconds, can solve *at most* around 500 problems. At a time budget of 1000 seconds, the differences are more marked, and the worst solved around 1600 problems and the best around 2000.

2 CDF plots

An alternative is (still after sorting the t_i) to plot $(t_1, 1)$, $t_2, 2)$ etc., and in general (t_k, k). If we normalise the y axis to $[0, 1]$ (*not* discarding the timeouts) we have an approximation to the cumulative distribution function. See [2] for some examples. Hence from these plots we can ask questions like "what is the probability of solving a random problem in t seconds".

3 Virtual Best Solver

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 is always equal to the time of some 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 to get a feeling for the variability between solvers.

We have therefore added this to our solvers on the diagrams,and counted how often a solver is the VBS. A variation on counting is provided by [1], 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 a one-second time-out".[1] EdN:1

4 Case Studies

[2] EdN:2

4.1 Predictable

For the first three tests we used a vector of baseline times (notionally in seconds) of
`cat(2,[1.1:0.002:2],[2:1:50],[50:5:300]);` in MatLab speak, i.e. 1.1 to 2 in steps of 0.002, 2 to 50 in step of 1, and 50 to 300 in steps of 5. We first measure four solvers: baseline, baseline less 1 second, 40% of baseline and a hybrid of 70% of (baseline less 0.5 seconds). The results are shown in Figure 2. "1 second faster" was quickest 284 times, and "60% faster" 267 times. However, "60% faster" took 48.3 seconds longer than the Virtual Best (which took 4311 seconds),"1 second faster" 6036 seconds longer, hybrid 3125 seconds longer and the baseline 6572 seconds longer.

4.2 Predictable plus Fuzz

What happens if we multiply each time by a random variable, uniform in [0.8,1.2]? 40 runs of this experiment give a mean VBS time of 4299 seconds, with a standard deviation of 53.455. In the counts of how often each solver was VBS, hybrid appeared, showing up between 5 and 17 times, with corresponding adjustments to the others. "1 second faster" was always the most common, with the ratio of it over "60% faster" ranging from 1.09 to 1.34. The plots (linear and semilogx) are in Figure 3.

4.3 Predictable plus Random

To the previous solvers, we add a "joker", that, on one problem in 10, takes 10% as long as the baseline. The results are shown in Figure 4. The joker was quickest 55 times, "1 second faster" was quickest 256 times, and "60% faster" 240 times. The time differences are that "60% faster" took 402 seconds longer

[1] EdNote: @All: do we want to discuss this? For example, this is just as subject to random fluctuation as the original, but in a different place. How about scoring "VBS points": 1 if VBS, 0 if more than 5% slower than VBS, and linear in between? Or any other smooth idea?

[2] EdNote: Alberto asked "what's the point"? Good question. I think the first two sections stand as a base case. Then see subsection "New".

than the Virtual Best, "1 second faster" 6390 seconds, hybrid 3479 seconds, the joker 5865 seconds and the baseline 6941 seconds.

4.4 Normal Distribution of times

It is far from clear what sort of distribution the running times of SAT, and even less SMT, solvers have, but it would be foolish to ignore the normal distribution. We took 36 hypothetical cases, with 9 different running times $t_0 = 1, \ldots, 9$, and standard deviations $t_0/10, 2t_0/10, 3t_0/10, 4t_0/10$ for each running time.

We used five (hypothetical) possible solvers. The base line one just took a time t at random from the relevant normal distribution: $t \in N(t_0, kt_0/10)$. The second one ran two copies in parallel, terminating when the first one did, the third ran three copies, and the next two 10 and 20. The data are in Figure 5.

It is also not clear whether we should assume t or $\log t$ is normally distributed. Hence we re-ran these experiments, but applied the normla distribution in $\log t$-space. The standard deviation was scaled so that it bore the same ratio to the mean as before. The data are in Figure 6.

4.5 Uniform

We then conisdered uniform distributions, with the lower bounds being $t_0 = 1, \ldots, 10$, and the upper bounds being 10 times that. We used the same hypothetical solvers as before. The data are in Figure 7.

Again, one could say that we should be uniform in $\log(t)$, and we did these computations. The data are in Figure 8.

4.6 Judgement

The data used so far had 500 "fast" problem ($<$ 2 seconds), 50 "medium" (between 2 and 50) and 5 hard (over 50). What happens if, instead, we have equal numbers in each bracket. The results from this, otherwise using the same methodology as section 4.3, are in Figure 9: the reader can see the difference from Figure 4: the current figure is dominated by the slow problems. The joker was quickest 593 times, "1 second faster" was quickest 256 times, and "60% faster" 5084 times.

5 Martin Brain adds

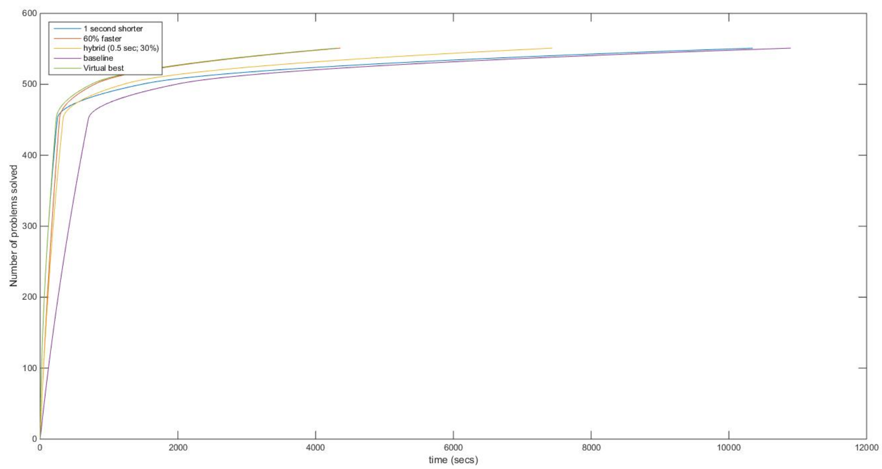
1. Scatter plots are used to compare pairs of solvers. For each benchmark you plot (sometimes using different colours or marks for SAT and UNSAT) a point with x location the time taken by solver 1 and y the time taken by solver 2. To make things easier to follow, people commonly add the diagonal (sometimes annotated with "solver 1 is faster" and "solver 2 is faster" on the relevant sides / corners) and the time-out lines.

References

[1] M. Janota, I. Lynce, and J. Marques-Silva. Algorithms for computing backbones of propositional formulae. *AI Communications*, 28:161–177, 2016.

[2] 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.

Figure 2: Data from Section 4.1



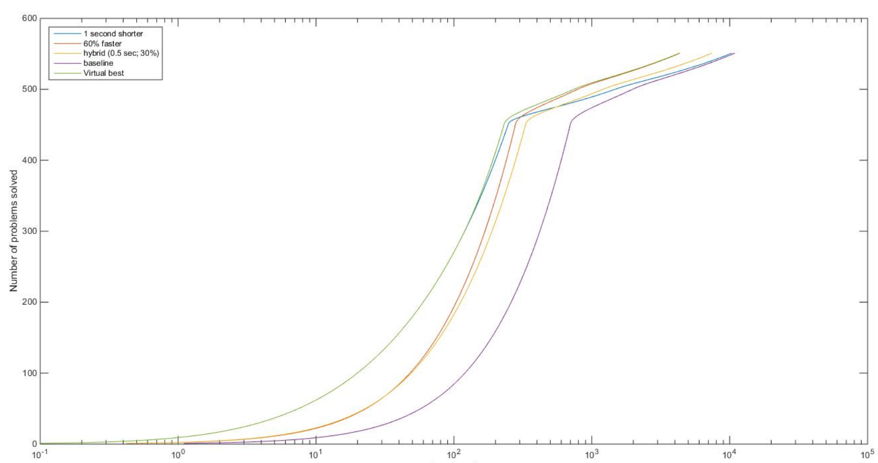
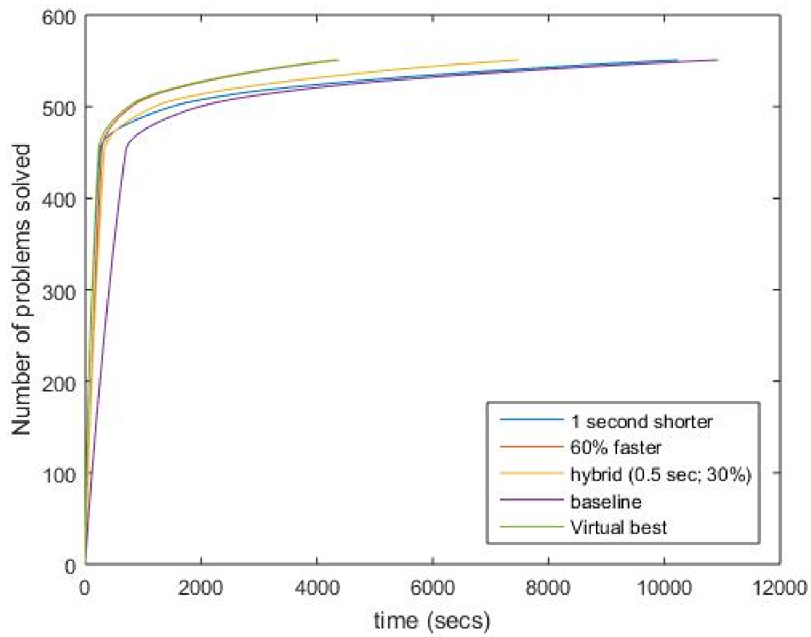


Figure 3: Data from Section 4.2



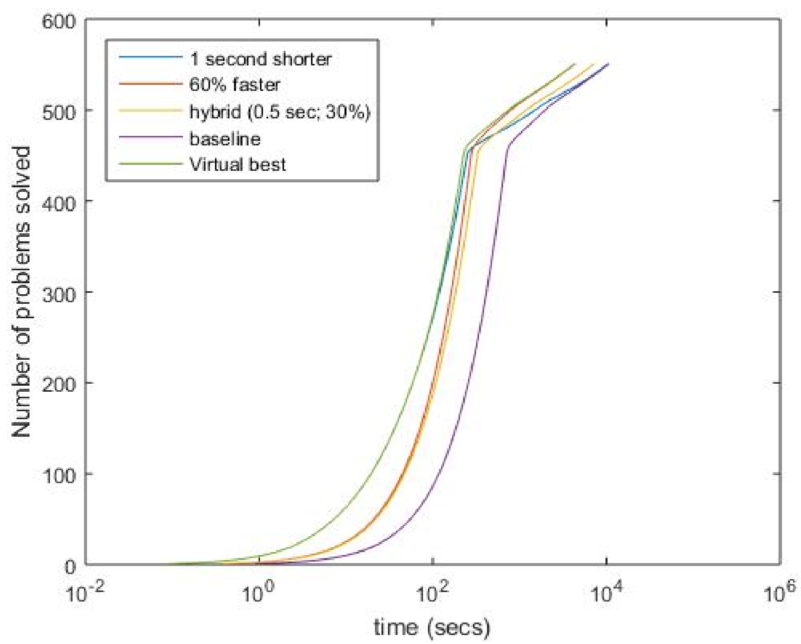
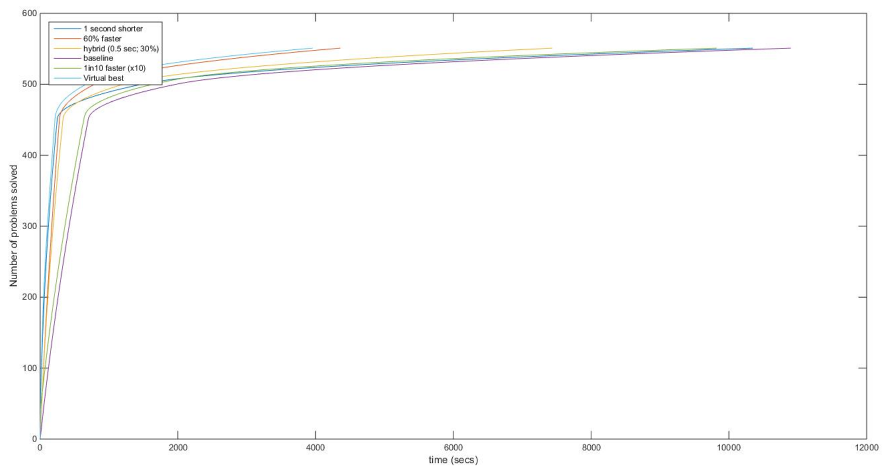


Figure 4: Data from Section 4.3



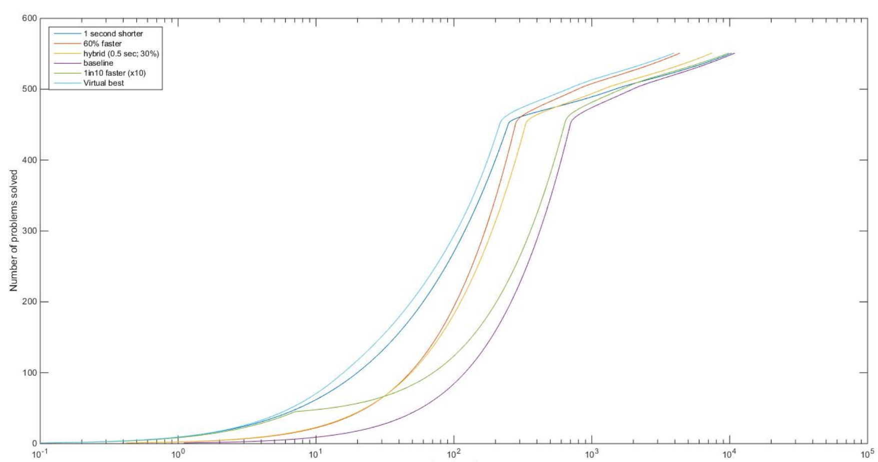


Figure 5: Data from Section 4.4 — Normal distribution

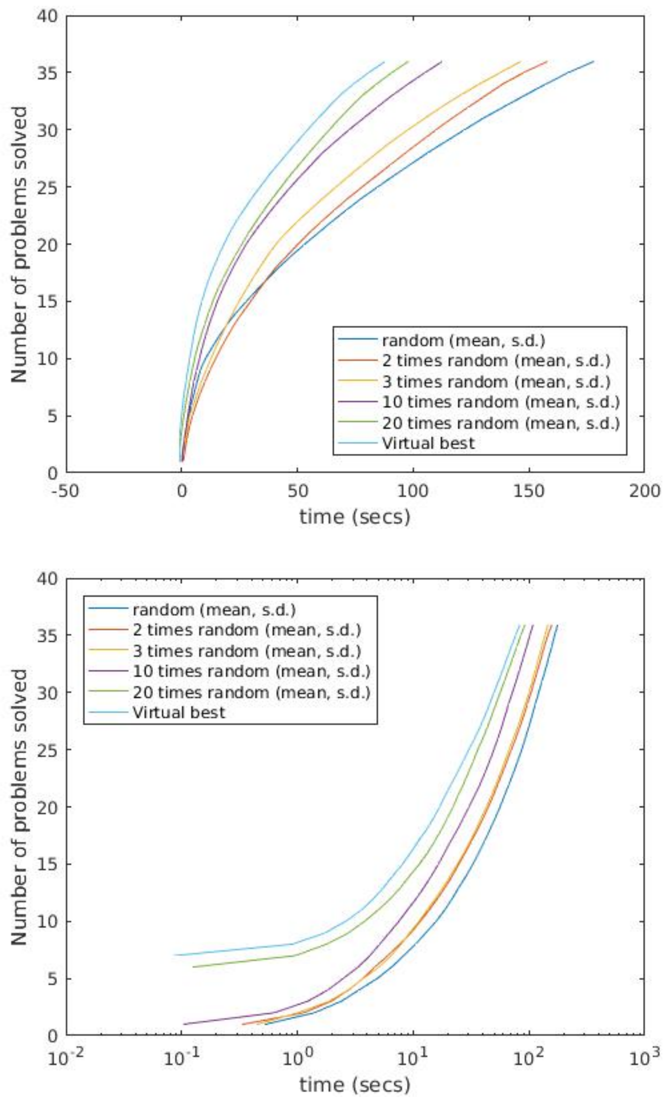


Figure 6: Data from Section 4.4 — log Normal distribution

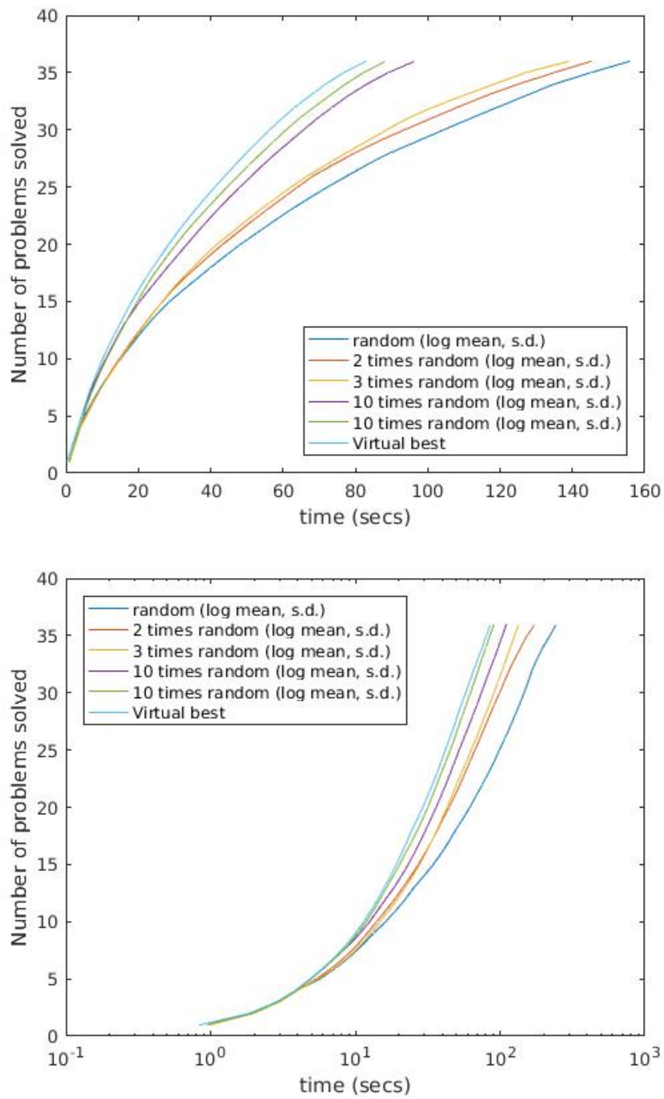
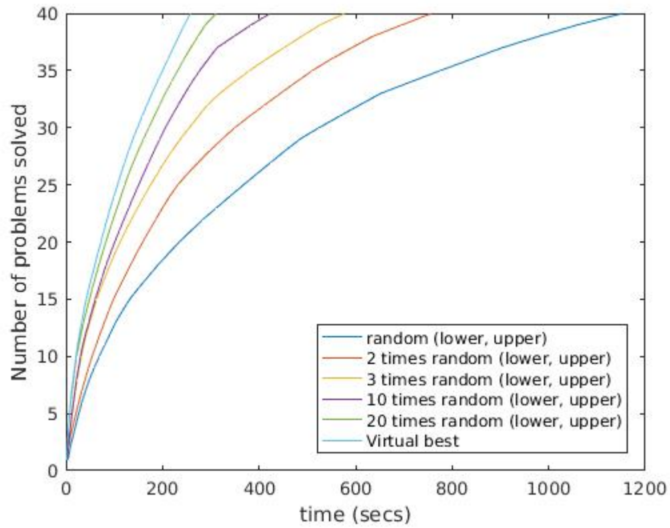


Figure 7: Data from Section 4.5 — Uniform distribution



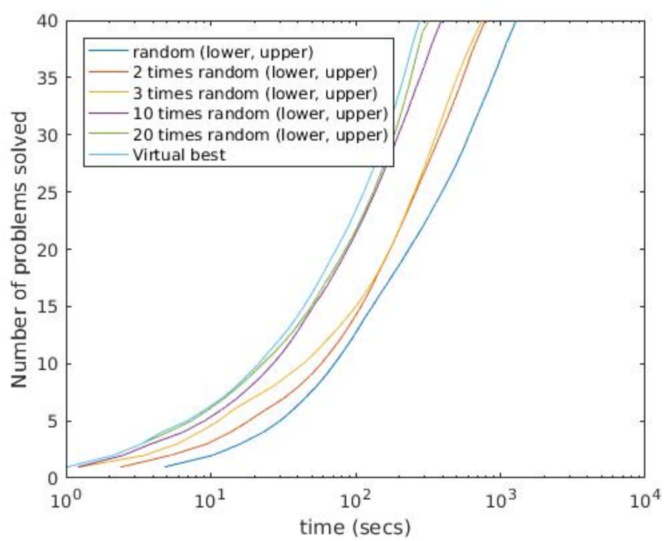
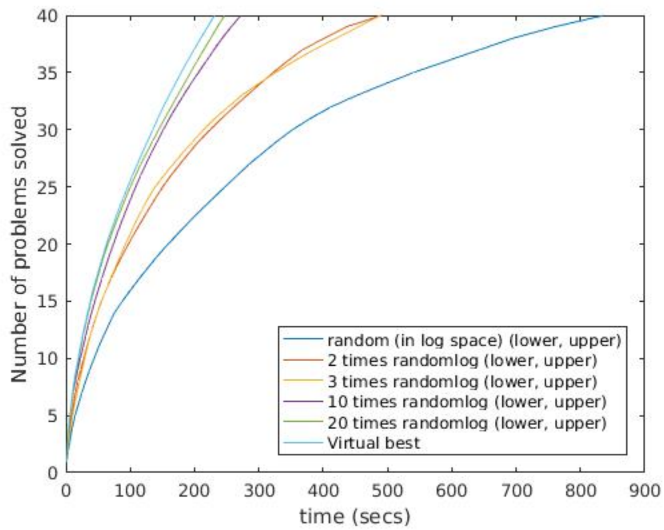


Figure 8: Data from Section 4.5 — log Uniform distribution



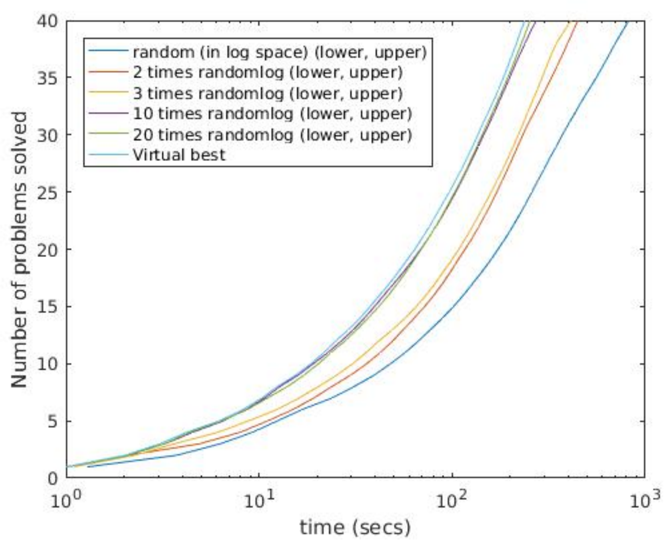


Figure 9: Data from Section 4.6