Competitive Evaluation of QBF Solvers: noisy data and scoring methods

Massimo Narizzano and Luca Pulina and Armando Tacchella

DIST, Università di Genova, Viale Causa, 13 – 16145 Genova, Italy
{narizzano,pulina,tacchella}@.dist.unige.it

Abstract. The 2006 evaluation of QBF solvers (QBFEVAL'06) is an automated reasoning competition involving systems for deciding QBFs. Although QBFEVAL'06 is the fourth in a series of events, it is the first competitive one, and also the one attracting the highest number of participants so far. QBFEVAL'06 is meant to designate a winner, but, at the same time, it should also summarize the performances of the solvers correctly, and provide some scientific insight about the current state of the art in QBF reasoning. In order to fulfill such requirements we had to deal with several research issues in the design of QBFEVAL'06, and, among them, two that can be of more general interest: noisy CPU time data, and the scoring methods used to compute the final ranking of the solvers. In this paper we investigate the above two issues. First, we provide an empirical characterization of CPU time noise, and we devise a model to deal with errors in CPU time measures efficiently, i.e., without having to run a solver several times on the same instance. Second, taking into account the noise model, we evaluate several scoring methods, including a new one that we devised to overcome the limitations of the others. Our method turns out to be a good compromise among competing requirements, and to be robust with respect to changes in the competition settings.

1 Introduction

The 2006 evaluation of QBF solvers (QBFEVAL'06) is an automated reasoning competition involving systems for deciding QBFs. Although QBFEVAL'06 is the fourth in a series of events (see [1–3] for previous QBFEVAL reports) , it is the first competitive one, and also the one attracting the highest number of participants so far. The automated reasoning research community has grown accustomed to competitive events like QBFEVAL'06. A non-exhaustive list of such sister events includes the CADE ATP System Competition (CASC) [4] for theorem provers in first order logic, the SAT Competition [5] for propositional satisfiability solvers, the International Planning Competition (see, e.g., [6]) for symbolic planners, the CP Competition (see, e.g., [7]) for constraint programming systems, and the Satisfiability Modulo Theories (SMT) Competition (see, e.g., [8]) for SMT solvers. The main purpose of the above events, including QBFEVAL'06, is to designate a winner. Even if such perspective can be limiting, and the results of automated reasoning systems competitions may provide less insight than controlled experiments in the spirit of [9], there is a general agreement that competitions raise interest in the community and play a fundamental role in the advancement of the state of the art, helping to set research challenges for developers and assess the current technological frontier for users. In order to (try to) meet the above requirements, the results of competitive events should also summarize the performances of the solvers as accurately as possible, and provide some scientific insight about the current state of the art in the automated reasoning branch of concern. In trying to pursue such goals in the design of QBFEVAL'06, we had to deal with several research issues, and, among them, two that can be of more general interest: noisy CPU time data, and the scoring method used to compute the final ranking of the solvers.

The issue of errors in CPU time measures occurred to us while designing QBFEVAL'06 (see [10] for a preliminary report). The phenomenon has to do with the inherent inaccuracy of CPU time measures performed by most operating systems, including QBFEVAL'06 platform. While such inaccuracy could be negligible for the casual user, in the context of a competition it is not. An assertion of the sort "solver A is better than solver B because the run-time of A is less than the run-time of B" should be based on intrinsic merit of A over B, and not on random fluctuations in the CPU time measures. When dealing with errors, the first question to ask is whether they are random errors affecting the measure, or systematic drifts from the correct measure: while the former ones can be dealt with, the latter ones must be eliminated. The second question to ask, should the answer to the first one be positive, is what it makes for a good model of the observed noise. In our case, while the answer to the first question is positive, i.e., CPU time drifts are indeed random, the answer to the second one is elusive, mainly because the observed run-times do not follow standard normal or log-normal distributions, even under simplifying assumptions. However, we show that standard approximations allow us to estimate with reasonable confidence the error to be taken into account when comparing the solvers, without having to run each solver several times on each instance.

Our analysis of scoring methods considers two different sets of them. The first set is comprised of methods used in automated reasoning systems contests, namely CASC [4], the SAT competitions [5], the (past) QBF evaluations [11], and a new method called YASM ("Yet Another Scoring Method") that we are evaluating as a candidate scoring method for QBFEVAL'06. The second set of scoring methods is comprised of procedures based on voting systems, namely Borda count [12], range voting [13] and Schulze's method [14]. The main difference between the methods of the first set is that using the methods of the second set amounts to consider the solvers as candidates and the problem instances as voters. Each voter ranks the candidates, i.e., the solver with the best performance on the instance is the preferred candidate, and all the other solvers are ranked accordingly. Finally, the votes are pooled to elect the winner of the contest. Our results show that YASM provides a good compromise when considering some measures that quantify desirable properties of the scoring procedures. In particular, the measures we propose account for:

- the degree of fidelity of the scoring methods, i.e., given a synthesized set of raw data, evaluate whether a scoring method distorts the results;
- the degree of stability of each scoring method with respect to perturbations (i) in the size of the test set, (ii) in the amount of resources available (CPU time), and (iii) in the quality of the test-set;
- the representativeness of each scoring method with respect to the state of the art expressed by the competitors.

We compute the above measures using part of the results from QBFEVAL'05 [3] and applying the noise model outlined above.

This paper builds on and extends previous work by one of the authors [10] in several ways. First, we analyze noise in CPU time measures and extract a model that is used to compare the scoring methods, while in [10] CPU times were assumed to be exact. Moreover, the version of YASM that we present here is new and improves on the one presented in [10]. The new YASM features a simpler calculation, and it is also more effective when compared to the old YASM and the other scoring methods. Finally, the comparison of scoring methods is broadened by the addition of new effectiveness measures (fidelity, see Section 5), and deepened by taking into account CPU time noise and an improved definition of SOTA relevance (see Section 5).

The paper is structured as follows. In Section 2 we introduce the case study of QBFEVAL'05 [3], we outline the working hypotheses underlying our analysis, and we introduce some state-of-the-art scoring methods. In Section 3 we propose a model for dealing with noisy CPU time measures. In Section 4 we introduce our new scoring method, and then we compare it with other methods in Section 5 using several effectiveness measures. We conclude the paper in Section 6 with a discussion about the current status of our research agenda. The appendices A and B contains tables, figures and plots that are referenced throughout Section 3 and Section 5, respectively.

2 Preliminaries

2.1 QBFEVAL'05

QBFEVAL'05 [3] is the third in a series of non-competitive events that preceded QBFEVAL'06. QBFEVAL'05 accounted for 13 competitors, 553 QBFs and three QBF generators submitted. The test set was assembled using a selection of 3191 QBFs obtained considering the submissions and the instances archived in QBFLIB [15]. The results of QBFEVAL'05 can be listed in a table RUNS comprised of four attributes (column names): SOLVER, INSTANCE, RESULT, and CPUTIME. The attributes SOLVER and INSTANCE report which solver is run on which instance. RESULT is a four-valued attribute: SAT, i.e., the instance was found satisfiable by the solver, UNSAT, i.e., the instance was found unsatisfiable by the solver, TIME, i.e., the solver exceeded a given time limit without solving the instance (900 seconds in QBFEVAL'05), and FAIL, i.e., the solver aborted for some reason (e.g., a run-time error, an inherent limitation of the solver, or any other reason beyond our control). Finally, CPUTIME reports the CPU time spent by the solver on the given instance, in seconds. In the analysis herewith presented we used a subset of QBFEVAL'05 RUNS table, including only the solvers that passed to the second stage of the evaluation, and the QBFs coming from classes of instances having fixed structure (see [3] for more details). Under these assumptions, RUNS table reduces to 4408 entries, one order of magnitude less than the original one. This choice allows us to disregard correctness issues, to reduce considerably the overhead of the computations required for our analysis, and, at the same time, maintain a significant number of runs.

2.2 Working hypotheses

The scoring methods that we evaluate, the measures that we compute and the results that we obtain, are based on the assumption that a table identical to RUNS as described above is the only input required by a scoring method. As a consequence, the scoring methods (and thus our analysis) do not take into account (i) memory consumption, (ii) correctness of the solution, and (iii) "quality" of the solution. As for (i), it turns out that measuring and comparing memory consumption is a tough exercise during a contest, mainly for two reasons. First, there are several definitions of memory consumption, e.g., peak memory usage as opposed to the total number of bytes in memory read/write operations, and it is not clear which one should be preferred. Second, it is complicated to measure memory consumption for systems as black boxes, which is usually the only view that the organizers of the contest have. Regarding (ii), checking the correctness of the solution is desirable for most automated reasoning systems. Unfortunately, the size of the certificate can be prohibitive in practice, and this is precisely the case of certificates for QBF (un)satisfiability which is a PSPACE-complete problem. Producing reasonably sized certificates, i.e., small proofs, from QBF solvers is still mainly a research issue (see, e.g., [16, 17]), and thus we do not have any certificates associated with QBFEVAL'05 RUNS table. Finally, (iii) matters only if there is indeed some solution on which quality indicators can be computed for the sake of comparison. No such indicators can be computed for simple SAT/UNSAT results, and the solutions, i.e., the proofs, are not currently available.

2.3 State of the art scoring methods

In the following we describe in some details the state of the art scoring methods used in our analysis. For each method we describe only those features that are relevant for our purposes. Further details can be found in the references provided.

CASC [4] The CASC scoring method applied to our setting yields the following guidelines. Solvers are ranked according to the number of problems solved, i.e., the number of times RESULT is either SAT or UNSAT. In case of a tie, the solver faring the lowest average on CPUTIME fields over the problems solved is preferred.

QBF evaluation [11] QBFEVAL scoring method is the same as CASC, except that ties are broken using the sum of CPUTIME fields over the problems solved.

SAT competition [5] The last SAT competition uses a *purse-based method*, i.e., the score of a solver on a given instance, is obtained by adding up three purses:

- the solution purse, which is divided equally among all solvers that solve the problem;
- the speed purse, which is divided unequally among all the competitors that solve the problem, first by computing the speed factor $F_{s,i}$ of a solver s on a problem instance i:

$$F_{s,i} = \frac{k}{1 + T_{s,i}} \tag{1}$$

where k is an arbitrary scaling factor (we set $k = 10^4$ according to [18]), and $T_{s,i}$ is the time spent by s to solve i; then by computing the speed award $A_{s,i}$, i.e., the portion of speed purse awarded to the solver s on the instance i:

$$A_{s,i} = \frac{P_i \cdot F_{s,i}}{\sum_r F_{r,i}} \tag{2}$$

where r ranges over the solvers, and P_i is the total amount of the speed purse for the instance i.
– the series purse, which is divided equally among all solvers that solve at least one problem in a given series (a series is a family of instances that are somehow related, e.g., different QBF encodings for some problem in a given domain).

The overall score of a solver is just the sum of its scores on all the instances of the test set, and the winner of the contest is the solver with the highest sum.

Borda count [12] Suppose that n solvers are participating to the contest. Each voter (instance) ranks the candidates (solvers) in ascending order considering the value of the CPUTIME field. Let $p_{s,i}$ be the position of a solver s in the ranking associated with instance i ($1 \leq p_{s,i} \leq n$). The score of s computed according to Borda count is just $S_{s,i} = n - p_{s,i}$. In case of time limit attainment and failure, we default $S_{s,i}$ to 0. The total score S_s of a solver s is the sum of all the scores, i.e., $S_s = \sum_i S_{s,i}$, and the winner is the solver with the highest score.

Range voting [13] The ranking position $p_{s,i}$ of each solver s on each instance i is computed as in Borda count. Then an arbitrary scale is used to associate a weight w_p with each of the n positions and the score $S_{s,i}$ is computed as $S_{s,i} = w_p \cdot p_{s,i}$ (default to 0 in case of time limit attainment or failure) and $S_s = \sum_i S_{s,i}$, as in Borda count. To compute w_p in our experiments we use a geometric progression with a common ratio $r = 2$ and a scale factor $a = 1$, i.e., $w_p = ar^{n-p}$ with $1 \leq p \leq n$.

Schulze's method [14] It is a pairwise voting method that enjoys the *Condorcet property*, i.e., the winner (resp. the loser) is guaranteed to beat (resp, lose to) every other solver in pairwise comparison. Borda count enjoys only the Condorcet property on the loser side, while Range voting does not enjoy such property. Schulze's method works as follows. A $n \times n$ square matrix M is computed, where the entries $M_{s,t}$ with $s \neq t$ account for the number of times that solver s is faster than t on some instance, while $M_{s,t} = 0$ if $s = t$. No points are awarded in case of ties, time limit attainment and failure. The score S_s according to which a solver s is ranked is computed as $S_s = \sum_t M_{s,t}$.

3 Noisy CPU time data

Given the setting described in Subsection 2.2, the only measures of merit at our disposal are the number of problems solved and the CPU times. The number of problems solved is correct as long as the CPU time measure used to enforce the time limit is so. Therefore, it turns out that to ensure accuracy of the results in our setting it is very important to tame potential sources of errors in the CPU time measures. In [10], it is assumed that

CPU times are unaffected by errors and such assumption is listed as a working hypothesis. If this assumption were true, repeated runs of a *deterministic* algorithm on the same instance should always yield the same quantity. However, as noticed also in [10], this is indeed not true on the current QBFEVAL platform.[1]

In order to study the influence of errors on CPU time measures, we used an implementation of the function PROBE presented in Figure 2 (Appendix A). The task accomplished by PROBE is quite simple: multiply two square matrices filled with random integers. We came up with this model considering that QBF solvers are CPU-intensive programs that rarely use floating point calculations, and that perform most of their computations on data stored in tabular format. Since most QBF solvers are also memory-intensive programs, we used dynamic memory allocation in PROBE in order to obtain a simple, yet realistic enough model.[2] Notice that we are not using QBFEVAL'05 solvers in this analysis since all of them are available to us as black boxes, they show different behaviors on the same instance, and their complex structure makes difficult to tune their expected run-time with sufficient precision. On the other hand, looking at Figure 2 we see that the size DIM of the multiplied matrices is the only parameter that can be used to tune the expected run-time of PROBE, which is monotonically increasing with DIM. The CPU time statistics of 100 runs of PROBE for expected run-times in the set $\Gamma = \{1, 5, 10, 20, 40, 50, 100, 500, 900\}$ are reported in Table 3 (Appendix A). The choice of such run-times is motivated by the fact that at least 85% of the problems solved by each solver in QBFEVAL'05 are processed in at most 50 seconds. In the following, we call N-sequence each sequence corresponding to some expected run-time $N \in \Gamma$ of PROBE. Inspection of Table 3 reveals that even for such a simple program as PROBE, notwithstanding the light-load conditions under which the data were obtained, there are fluctuations in the observed run-times that can make the comparisons between CPU times unreliable. As we will shortly see, the following facts are also true:

1. The errors in CPU time measures are random, i.e., there seems to be no systematic drift that can (and should) be eliminated a priori in the N-sequences.
2. The distribution of the observed run-times does not follow a standard, i.e., normal (or log-normal), distribution, unless the run-time is relatively high (around 900 seconds).
3. The statistics that measure the spread, i.e., standard deviation and interquartile range, of the observed run-times are not constant, but they appear to grow together with the measures of center, i.e., the mean and the median, of the observed run-times.

We now analyze each of these facts in turn using the data summarized in Table 3 and, in the end, we propose an approximated model to deal with the CPU time noise.

To confirm fact (1) above, we can resort to autocorrelation plots for each of the N-sequences. Autocorrelation plots, see Figure 3 (Appendix A), are obtained by computing autocorrelations for the observed run-times in a given N-sequence at varying

[1] A farm of identical 3GHz PIV PCs with 1GB of main memory, running `Debian GNU/Linux` (`sarge` distribution with a 2.4.x kernel)

[2] It can be observed that PROBE does not deallocate memory dynamically. However, although QBF solvers can in principle perform allocations *and* deallocations, in our experience their memory consumption is rarely found to be substantially decreasing during their computation.

time lags. If the sequence is truly random, such autocorrelations should be near zero for any and all time-lag separations (except, of course, time-lag 0 which is 1 by definition). If non-random, then one or more of the autocorrelations will be significantly non-zero, where "significantly" here means outside a reasonable confidence interval (we use a 95% confidence interval in our analysis). As we can see in Figure 3, for each N-sequence, the corresponding autocorrelation plot does not show any particular pattern, and most of the correlation values are well within the confidence band. From this analysis, we can conclude that the N-sequences can be considered random, and thus devoid of systematic errors. Moreover, this also justifies the use of statistical tools and tests to analyze the sequences, which would not make sense otherwise.

We now turn to the question of the distribution characterizing each N-sequence. The results of this analysis are summarized in Figures 4 and 5 and in Table 4 (Appendix A). In Figure 4, for each N-sequence we provide: (i) a histogram describing the frequency distribution of the observed data; (ii) an estimate of the probability density function (pdf) based on the observations (red dots), and (iii) the pdf of a normal distribution having the same mean and standard deviation of the observed data (blue dots). Figure 4 tells us that most N-sequences yield a distribution of the observed run-times which is both skewed, i.e., not symmetric, and kurtotic, i.e., with a substantial number of observations that are far from the center of the distribution (a.k.a. "fat tails"). This is confirmed quantitatively by the skewness and kurtosis values reported in Figure 4 which are significantly different from zero in most cases. The only exception is the 900-sequence whose distribution is approximately normal (red and blue dots almost coincide). The suspect of non-normality for most of the N-sequences is confirmed in Figure 5, where normal quantile-quantile plots are shown. Should the observed data be normally distributed, the points in the scatter plots of Figure 5 would fall approximately on a straight line. Indeed this is almost always false, both considering all the observed data (blue line across the plots in Figure 5) and when excluding more extreme observations (red line in Figure 5). The only exception to this pattern is represented by the 900-sequence and, to a lesser extent, by the 500-sequence. A final confirmation of this trend comes from the p-values obtained performing the Shapiro-Wilk normality test and reported in the 4th (normality) and 5th (log-normality) columns of Table 4. The Shapiro-Wilk test [19] is a nonparametric test to assess normality. A test for log-normality can thus be obtained by applying the natural logarithm to the sequence under test. The null hypothesis H_0 of the test is that the observed values came from a normal distribution (the alternative hypothesis H_a states of course the contrary). Since the p-values obtained from the test are very small, we can safely reject H_0 and conclude that all the N-sequences are not normally (or log-normally) distributed, with the only exception of the 900-sequence that yields a sufficiently high p-value for normality to conclude that we cannot safely reject H_0 in this particular case. Wrapping up, we confirm fact (2) above, and conclude that the N-sequences most probably do not follow a normal (or log-normal) distribution, particularly for small values of N. For increasing values of N, the distribution becomes close to normal (see, e.g., the blue and red line in Figure 5 that get closer for increasing values of N).

Fact (3), can be appreciated by looking at the first two columns of Table 4 (Appendix A). It is quite clear that the larger the mean μ, the larger the standard deviation

σ. Quantitatively, this can be checked by performing Pearson's test for (linear) correlation strength between the values of μ and σ. The correlation coefficient thereby obtained is 99.68% and the p-value of the test is less than 0.001, indicating that we can safely reject H_0, i.e., the null correlation hypothesis.

Although fact (2) left us with no simple means to estimate average or maximum errors, we checked how many observations for each N-sequence fall in the $\mu \pm 3\sigma$ interval. If the distributions were normal, then 99.7% of the observations should fall in such interval. As we can see from column six in Table 4, this is indeed not true for the N-sequences, but nevertheless the percentage of observations that fall in the $\mu \pm 3\sigma$ interval is always a respectable 97% at least. Leveraging fact (3), we can thus proceed to obtain an error estimate as follows. We obtain $\hat{\sigma}(\mu) = a \cdot \mu + b$ by performing least squares regression, yielding estimated values of $a = 0.02$ (slope) and $b = -0.35$ (intercept). The confidence turns out to be high for the slope (the probability of the slope being 0 is less than 0.001), and more modest for the intercept (there is a 14% chance that the intercept is 0). We therefore set $b = 0$ and compute the estimate $\hat{\sigma}(\mu)$ of σ as $\hat{\sigma} = 0.02\mu$. As we can see from column seven of Table 4 this turns out to be an overestimation of the actual standard deviation σ particularly for small CPU times, but since we are looking for uncertainty on CPU times, overestimation is definitely safer. Indeed, considering Table 4 (last column) we can see that at least 98% of the observed values fall within the $\mu \pm 3\hat{\sigma}$ interval.

Summing up, when comparing run-times t_1 and t_2 in the scoring methods we conclude that they are different iff either $t_1 < t_2$ and $(t_1 + 0.06t_1) < (t_2 - 0.06t_2)$, or $t_1 > t_2$ and $(t_1 - 0.06t_1) > (t_2 + 0.06t_2)$; in all the other cases, t_1 and t_2 must be considered indistinguishable.

4 YASM: Yet Another Scoring Method (Revisited)

While the scoring methods used in CASC and QBF evaluations are straightforward, they do not take into account some aspects that are indeed considered by the purse-based method used in the last SAT competition. On the other hand, the purse-based method used in SAT requires some oracle to assign purses to the problem instances, so the results can be influenced heavily by the oracle. In [10] a first version of YASM was introduced as an attempt to combine the two approaches: a rich method like the purse-based one, but using the data obtained from the runs only. As reported in [10], YASM featured a somewhat complex calculation, yielding unsatisfactory results, particularly in the comparison with scoring methods based on voting systems. Here we revise the original version of YASM to make its computation simpler, and to improve its performance using ideas borrowed from voting systems. From here on, we call YASMv2 the revised version, and YASM the original one presented in [10]. YASMv2 requires a preliminary classification whereby a hardness degree H_i is assigned to each problem instance i using the same equation as in CASC [4] (and YASM):

$$H_i = 1 - \frac{S_i}{S_t} \tag{3}$$

where S_i is the number of solvers that solved i, and S_t is the total number of participants to the contest. Considering equation (3), we notice that $0 \leq H_i \leq 1$, where

	CASC	QBF	SAT	YASM	YASMv2	Borda	r.v.	Schulze
CASC	–	1	0.71	0.86	0.79	0.86	0.71	0.86
QBF		–	0.71	0.86	0.79	0.86	0.71	0.86
SAT			–	0.86	0.86	0.71	0.71	0.71
YASM				–	0.86	0.71	0.71	0.71
YASMv2					–	0.86	0.86	0.86
Borda						–	0.86	1
r. v.							–	0.86
Schulze								–

Table 1. Homogeneity of scoring methods.

$H_i = 0$ means that i is relatively easy, while $H_i = 1$ means that i is relatively hard. We can then compute the score $SB_{s,i}$ of a solver s on a given instance i (this definition changes with respect to YASM):

$$S_{s,i} = k_{s,i} \cdot (1 + H_i) \cdot \frac{L - T_{s,i}}{L - M_i} \tag{4}$$

where L is the time limit, $T_{s,i}$ is the CPU time used up by s to solve i ($T_{s,i} \leq L$), and $M_i = min_s\{T_{s,i}\}$, i.e., M_i is the time spent on the instance i by the *SOTA solver* defined in [3] to be the ideal solver that always fares the best time among all the participants. The hybridization with voting systems comes into play with the coefficient $k_{s,i}$ which is computed as follows. Suppose that n solvers are participating to the contest. Each instance ranks the solvers in ascending order considering the value of the CPUTIME field. Let $p_{s,i}$ be the position of a solver s in the ranking associated with instance i ($1 \leq p_{s,i} \leq n$), then $k_{s,i} = n - p_{s,i}$. In case of time limit attainment and failure, we default $k_{s,i}$ to 0, and thus also $S_{s,i}$ is 0. The total score of a solver S_s is just the sum of the scores obtained on the instances, i.e., $S_s = \sum_i S_{s,i}$.

We can see from equation (4) that in YASMv2 the score of a solver on a given instance is influenced by three factors, namely (i) a Borda-like positional weight ($k_{s,i}$), (ii) the relative hardness of the instance ($1 + H_i$), and (iii) the relative speed of the solver with respect to the fastest solver on the instance ($\frac{L-T_{s,i}}{L-M_i}$). Intuitively, coefficient (ii) rewards the solvers that are able to solve hard instances, while (iii) rewards the solvers that are faster than other competitors. The coefficient $k_{s,i}$ has been added to stabilize the scores and make them less sensitive to an initial bias in the test set. As we show in the next Section, this combination allows YASMv2 to reach the best compromise among different effectiveness measures.

5 Experimental Evaluation

5.1 Homogeneity

The rationale behind this measure (introduced in [10]) is to verify that, on a given test set, the scoring methods considered (i) do not produce exactly the same solver rankings, but, at the same time, (ii) do not yield antithetic solver rankings. Thus, homogeneity is not an effectiveness measure per se, but it is a preliminary assessment that we are performing an apple-to-apple comparison and that the apples are not exactly the same.

Method	Mean	Std	Median	Min	Max	IQ Range	F
QBF	182.25	7.53	183	170	192	13	88.54
CASC	182.25	7.53	183	170	192	13	88.54
SAT	87250	12520.2	83262.33	78532.74	119780.48	4263.94	65.56
YASM	46.64	2.22	46.33	43.56	51.02	2.82	85.38
YASMv2	1257.29	45.39	1268.73	1198.43	1312.72	95.11	91.29
Borda	984.5	127.39	982.5	752	1176	194.5	63.95
r. v.	12010.25	5183.86	12104	5186	21504	8096	24.12
SCHULZE	982.3	57.23	986	875	1072	76.5	81.62

Table 2. Fidelity of scoring methods. As far as **SAT** is concerned, the series purse is not assigned.

Homogeneity is computed as in [10] considering the Kendall rank correlation coefficient τ [20] which is a nonparametric coefficient best suited to compare rankings. τ is computed between any two rankings and it is such that $-1 \leq \tau \leq 1$, where $\tau = -1$ means perfect disagreement, $\tau = 0$ means independence, and $\tau = 1$ means perfect agreement. Table 1 shows the values of τ computed for the scoring methods considered, arranged in a symmetric matrix where we omit the elements below the diagonal (r.v. is a shorthand for range voting). Values of τ close to, but not exactly equal to 1 are desirable. Table 1 shows that this is indeed the case for the scoring methods considered using QBFEVAL'05 data. Only two couples of methods (QBF-CASC and Schulze-Borda) show perfect agreement, while all the other couples agree to some extent, but still produce different rankings.

5.2 Fidelity

We introduce this measure to check whether the scoring methods under test introduce any distortion with respect to the true merits of the solvers. Our motivation is that we would like to extract some scientific insight from the final ranking of QBFEVAL'06 and not just winners and losers. Of course, we have no way to know the true merits of the QBF solvers: this would be like knowing the true statistic of some population. Therefore, we measure fidelity by feeding each scoring method with "white noise", i.e., a table RUNS having the same structure outlined in Subsection 2.1 and filled with random results. In particular, we assign to RESULT one of SAT/UNSAT, TIME and FAIL values with equal probability, and a value of CPUTIME chosen uniformly at random in the interval [0;1]. Given this artificial setting, we know in advance that the true merit of the competitors is approximately the same. A high-fidelity scoring method is thus one that computes approximately the same scores for each solver, and thus produces a final ranking where scores have a small variance-to-mean ratio.

The results of the fidelity test are presented in Table 2. In the Table, each line contains the statistics of a scoring method, and the columns show, from left to right, the mean, the standard deviation, the median, the minimum, the maximum and the interquartile range of the scores produced by each scoring method when fed by white noise. The last column is our fidelity coefficient F, i.e., the percent ratio between the lowest score (solver ranked last) and the highest one (solver ranked first): the higher the value of F, the more the fidelity of the scoring method. As we can see from Table 2, the fidelity of YASMv2 is better than that of all the other methods under test, including

QBF and CASC which are second best, and have higher fidelity than YASM. Notice that range voting, and to a lesser extent also SAT and Borda methods, introduce a substantial distortion. In the case of range voting, this can be explained by the exponential spread that separates the scores, and thus amplifies even small differences.

5.3 RDT-stability and DTL-stability

Stability on a randomized decreasing test set (RDT-stability), and stability on a decreasing time limit (DTL-stability) have been introduced in [10] to measure how much a scoring method is sensitive to perturbations that diminish the size of the original test set, and how much a scoring method is sensitive to perturbations that diminish the maximum amount of CPU time granted to the solvers, respectively. The results of RDT- and DTL-stability tests are presented in the plots of Figures 6 and 7 (Appendix B). We obtained such plots using the CPU time noise model herewith introduced, and considering YASMv2 instead of YASM. However, the conclusion that we reach are the same of [10], and precisely:

- All the scoring methods considered are RDT-stable up to 400, i.e., a random sample of 151 instances is sufficient for all the scoring methods to reach the same conclusions that each one reaches on the heftier set of 551 instances used in QBFEVAL'05.
- Decreasing the time limit substantially, even up to one order of magnitude, is not influencing the stability of the scoring methods considered, except for some minor perturbations for QBF/CASC and SAT methods. Moreover, independently from the scoring method used and the amount of CPU time granted, the best solver is always the same.

Indeed, while the above measures can help us extract general guidelines about running a competition, in our setting they do not provide useful insights to discriminate the relative merits of the scoring methods.

5.4 SBT-stability

Stability on a solver biased test set (SBT-stability) is introduced in [10] to measure how much a scoring method is sensitive to a test set that is biased in favor of a given solver. Let Γ be the original test set, and Γ_s be the subset of Γ such that the solver s is able to solve exactly the instances in Γ_s. Let $R_{q,s}$ be the ranking obtained by applying the scoring method q on Γ_s. If $R_{q,s}$ is the same as the original ranking R_q, then the scoring method q is SBT-stable with respect to the solver s. Notice that, contrarily to what stated in [10], SBT-stability alone is not a sufficient indicator of the capacity of a scoring method to detect the absolute merit of the participants. Indeed, it turns out that a very low-fidelity method such as range voting is remarkably SBT-stable. This because we can raise the SBT-stability of a scoring by decreasing its fidelity: in the limit, a scoring method that assigns fixed scores to each solver, has the best SBT-stability and the worst fidelity. Therefore, a scoring method showing a high SBT-stability is relatively immune to bias in the test set, but it must also feature a high fidelity if we are to conclude that the method provides a good hint at detecting the absolute merit of the solvers.

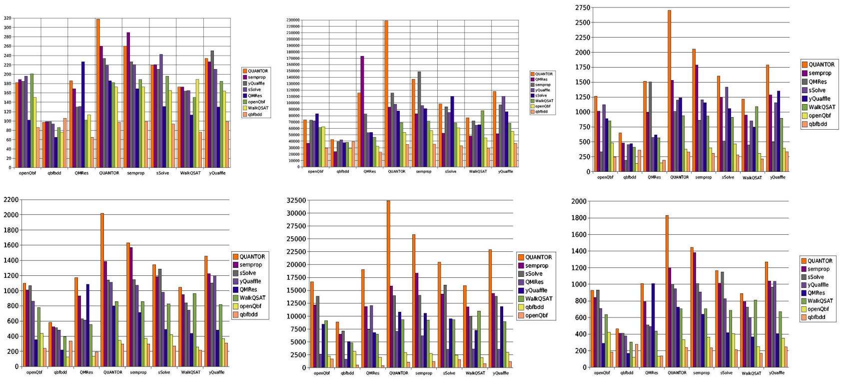

Fig. 1. SBT-stability plots.

Figure 1 shows the plots with the results of the SBT-stability measure for each scoring method considering the noise model and YASMv2 (the layout is the same as Figures 6 and 7 in Appendix B). The x-axis reports the name of the solver s used to compute the solver-biased test set Γ_s and the y-axis reports the score value. For each of the Γ_s's, we report eight bars showing the scores obtained by the solvers using only the instances in Γ_s. The order of the bars (and of the legend) corresponds to the ranking obtained with the given scoring method on the original test set Γ. As we can see from Figure 1 (top-left), CASC/QBF scoring methods are not SBT-stable: for each of the Γ_s, the original ranking is perturbed and the winner becomes s. Notice that on Γ_{QUANTOR}, CASC/QBF yield the same ranking that they output on the complete test set Γ. The SAT competition scoring method (Figure 1, top-center) is not SBT-stable, not even on the test set biased on its alleged winner QUANTOR. YASMv2 is better than both CASC/QBF and SAT, since its alleged winner QUANTOR is the winner on biased test sets as well. Borda count (Figure 1, bottom-left) is not SBT-stable with respect to any solver, but the alleged winner (QUANTOR) is always the winner on the biased test sets. Moreover, the rankings obtained on the test sets biased on QUANTOR and SEMPROP are not far from the ranking obtained on the original test set. Also range voting (Figure 1, bottom-center), is not SBT-stable with respect to any solver, but the solvers ranking first and last do not change over the biased test sets. Finally, Schulze's method (Figure 1, bottom-right) is SBT-stable only with respect to its alleged winner QUANTOR.

Looking at the results presented above, we can see that YASMv2 performance in terms of SBT stability lies in between classical automated reasoning contests methods and methods based on voting systems. This fact is highlighted in Table 5 (Appendix B), where for each scoring method we compute the Kendall coefficient between the ranking obtained on the original test set Γ and each of the rankings obtained on the Γ_s test sets, including the mean coefficient observed. Overall, YASMv2 turns out to be, on average, better than CASC/QBF, SAT, and YASM, while it is worse, on average, than the methods based on voting systems. However, if we consider also the results of Table 2 about fidelity, we can see that YASMv2 offers the best compromise between SBT-stability and fidelity. Indeed, while CASC/QBF methods have a relatively high fidelity, they perform poorly in terms of SBT-stability, and SAT method is worse than YASMv2 both in terms of fidelity and in terms of SBT-stability. Methods based on voting systems are all more SBT-stable that YASMv2, but they have poor fidelity coefficients, with the only exception of Schulze's method whose fidelity coefficient is within 10% of YASMv2 coefficient. We consider this good performance of YASMv2 a result of our choice to hybridize classical methods used in automated reasoning contests and methods based on voting systems. This helped us to obtain a scoring method which is less sensitive to bias, and, at the same time, a good indicator of the absolute merit of the competitors.

5.5 SOTA-relevance

This measure was introduced in [10] to understand the relationship between the ranking obtained with a scoring method and the strength of a solver, as witnessed by its contribution to the SOTA solver. As mentioned in Subsection 4, the SOTA solver is the ideal solver that always fares the best time among all the participants. Indeed, a participant contributes to the SOTA solver whenever it is the fastest solver on some instance. In [10] SOTA-relevance was obtained by counting the number of such events for any given solver, and then computing the Kendall coefficient between the ranking thereby induced and the ranking obtained with any given scoring method. However, it turns out that evaluating the SOTA-contribution of each solver by simply counting the number of times that it is faster than other solvers can be misleading. To understand this, consider the following example. Suppose that a solver A solves 50% of the test set using time *at most* t_A and times out on the rest, and that solver B, on the contrary, solves all the problems where A times out using time *at most* t_B but it does time out on the problems that A solves. Finally, suppose that a solver C is able to solve all the problems in the test set using time *at least* t_C where $t_C > t_A$ and $t_C > t_B$. Given our definition of SOTA solver, it turns out that C is never contributing to it. Evaluating the SOTA contribution using a simple count as described in [10] would induce a ranking where C is last. However, C is, on average, better than both A and B and this will probably be correctly spotted by high-fidelity methods, which would turn out to have a very low SOTA-relevance.

In order to overcome the above problem we redefine here SOTA-relevance in terms of SOTA-distance. SOTA-distance is the distance metric obtained by computing the Euclidean norm between the CPU times of any given solver and the SOTA solver. The resulting values of the metrics induce a ranking that can be used to compute the Kendall coefficient yielding the SOTA-relevance. Table 6 (Appendix B) shows the values of the coefficients thereby obtained for each scoring method. Notice that according to our new definition of SOTA-relevance, CASC/QBF methods turn out to have the highest such relevance possible, i.e., $\tau = 1$. Therefore the other coefficients correspond to the first row of Table 1 about homogeneity results. Notice that YASMv2 has a better SOTA relevance than SAT and range voting, but worse than all the other methods, including YASM. Given the positive results of YASMv2 insofar fidelity and SBT-stability are concerned, we consider this an indication that the SOTA solver abstraction, despite its relevance for practical purposes, might not be a good indicator of the merit of the solvers.

6 Conclusions

Summing up, the analysis presented in this paper allowed us to make substantial progress in the research agenda of QBFEVAL'06. In [10], modeling the CPU time noise and improving YASM were cited as future directions. In this paper we have presented a thorough analysis of the CPU time noise that allowed us to reach some understanding of the phenomenon, and to come up with a rough but effective model that deals with errors in CPU time measures. We have also improved YASM with YASMv2, which features a simpler calculation, yet it is more powerful than YASM in terms of SBT-stability and fidelity. Our empirical evaluation tools of scoring methods have also improved with the addition of the noise model, the fidelity measure and the improved definition of SOTA-relevance. We confirmed some of the conclusions reached in [10], namely that independently of the specific scoring method used, a larger test set is not necessarily a better test set, and that a higher time limit does not necessarily result in a more informative contest. On the other hand, while scoring methods based on voting systems emerged from [10] as "moral" winners over other scoring methods, the analysis presented in this paper shows that better results can be achieved using hybrid techniques such as YASMv2.

The next items in our research agenda include two main issues. The first has to do with the analysis of the statistical significance of the rankings obtained in the spirit of [6] and, particularly, the relationship between the various scoring methods and the indications obtained using statistical hypothesis testing. The second, although not directly linked to the results herewith presented, is the investigation on computing efficient certificates for QBFs (un)satisfiability. This would allow us to relax further our working hypothesis and broaden the spectrum of our conclusions, since we could take into account correctness of the solvers and, possibly, the quality of the solutions.

References

1. D. Le Berre, L. Simon, and A. Tacchella. Challenges in the QBF arena: the SAT'03 evaluation of QBF solvers. In *Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, volume 2919 of *Lecture Notes in Computer Science*. Springer Verlag, 2003.
2. D. Le Berre, M. Narizzano, L. Simon, and A. Tacchella. The second QBF solvers evaluation. In *Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, Lecture Notes in Computer Science. Springer Verlag, 2004.
3. M. Narizzano, L. Pulina, and A. Tacchella. The third QBF solvers comparative evaluation. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:145–164, 2006. Available on-line at `http://jsat.ewi.tudelft.nl/`.
4. G. Sutcliffe and C. Suttner. The CADE ATP System Competition. `http://www.cs.miami.edu/~tptp/CASC`.
5. D. Le Berre and L. Simon. The SAT Competition. `http://www.satcompetition.org`.
6. D. Long and M. Fox. The 3rd International Planning Competition: Results and Analysis. *Artificial Intelligence Research*, 20:1–59, 2003.
7. M.R.C. van Dongen. Introduction to the Solver Competition. In *CPAI 2005 proceedings*, 2005.

8. C. W. Barrett, L. de Moura, and A. Stump. SMT-COMP: Satisfiability Modulo Theories Competition. In *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 20–23, 2005.
9. J. N. Hooker. Testing Heuristics: We Have It All Wrong. *Journal of Heuristics*, 1:33–42, 1996.
10. L. Pulina. Empirical Evaluation of Scoring Methods. In *Proceedings of STAIRS*, 2006. Accepted contribution. Available on line at `http://www.star.dist.unige.it/~pulina/paper/stairs06-Pulina.pdf` [2006-5-12].
11. M. Narizzano, L. Pulina, and A. Taccchella. QBF solvers competitive evaluation (QBFEVAL). `http://www.qbflib.org/qbfeval`.
12. D. G. Saari. *Chaotic Elections! A Mathematician Looks at Voting*. American Mathematical Society, 2001.
13. RangeVoting.org. `http://math.temple.edu/~wds/crv/`.
14. M. Schulze. A New Monotonic and Clone-Independent Single-Winner Election Method. *Voting Matters*, pages 9–19, 2003.
15. E. Giunchiglia, M. Narizzano, and A. Tacchella. Quantified Boolean Formulas satisfiability library (QBFLIB), 2001. `www.qbflib.org`.
16. Y. Yu and S. Malik. Verifying the Correctness of Quantified Boolean Formula(QBF) Solvers: Theory and Practice. In *ASP-DAC*, 2005.
17. M. Benedetti. Evaluating QBFs via Symbolic Skolemization. In *Eleventh International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2004)*, volume 3452 of *Lecture Notes in Computer Science*. Springer Verlag, 2004.
18. A. Van Gelder, D. Le Berre, A. Biere, O. Kullmann, and L. Simon. Purse-Based Scoring for Comparison of Exponential-Time Programs, 2006. Unpublished draft.
19. G.K. Kanji. *100 Statistica Tessts - New Edition*. SAGE Publications, 1999.
20. M. Kendall. *Rank Correlation Methods*. Charles Griffin & Co. Ltd., 1948. Reference available on line from `http://en.wikipedia.org/wiki/Rank_correlation`.

A Tables and figures of Section 3

```
1  PROBE( DIM:integer )
2    M, N, P: vector< vector< integer > >
3    M.resize(DIM)
4    N.resize(DIM)
5    for i ← 1 to DIM do
6      M[i].resize(DIM)
7      N[i].resize(DIM)
8      for j ← 1 to DIM do
9        M[i][j] = {a random number}
10       N[i][j] = {a random number}
11   P.resize(DIM)
12   for i ← 1 to DIM
13     P[i].resize(DIM);
14     for j ← 1 to DIM
15       P[i][j] ← 0
16       for k ← 0 to DIM
17         P[i][j] = P[i][j] + M[i][k] * N[k][j]
18   return P
```

Fig. 2. Probe to model CPU time noise. PROBE implements a matrix multiplication algorithm with dynamically sized matrices. The parameter DIM enables tuning of the expected CPU time. Dynamic allocation occurs in lines 5,6 (M and N row indexes), 8,9 (M and N row vectors), 13 (P row indexes) and 15 (P row vectors).

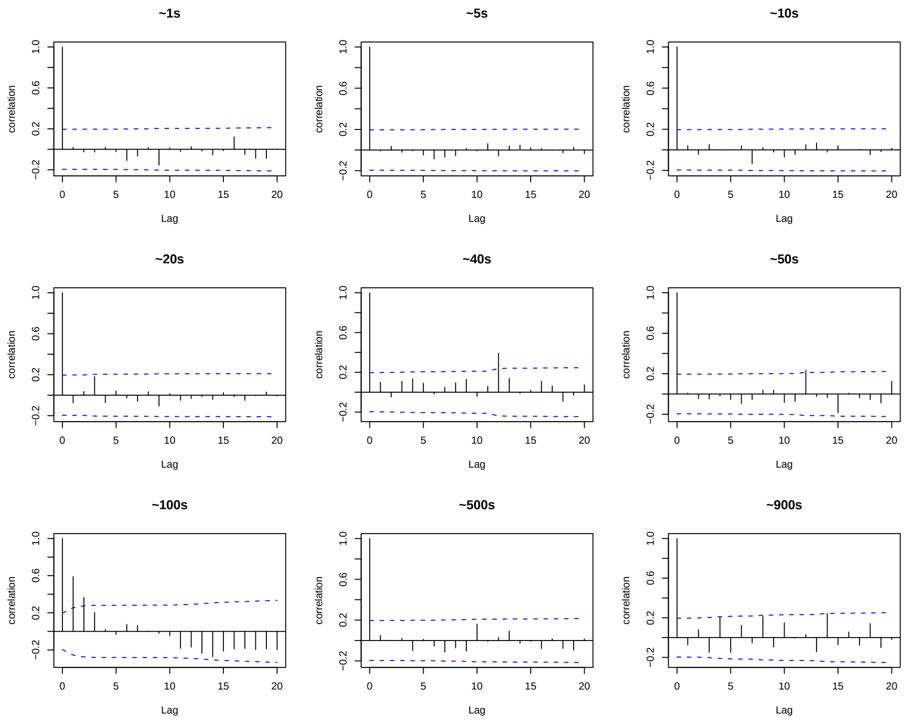

Fig. 3. Autocorrelation of repeated CPU time samples, for expected run-times of 1s, 5s, 10s (first row), 20s, 40s, 50s (second row), and 100s, 500s and 900s (third row). The Y axis reports the value of the autocorrelation coefficient (in the range [-1;1]) for increasing lags reported on the X axis. The dashed lines delimit 95% confidence intervals.

Run-time	Min	Q1	Median	Mean	Q3	Max	Std. dev.
~1	0.99	1.00	1.00	0.99	1.00	1.01	0.005
~5	4.91	5.10	5.10	5.10	5.11	5.13	0.021
~10	10.13	10.32	10.33	10.32	10.33	10.50	0.032
~20	21.56	21.74	21.75	21.75	21.76	22.08	0.051
~40	39.48	39.56	39.57	39.58	39.60	40.07	0.065
~50	52.31	52.41	52.50	52.57	52.66	53.59	0.220
~100	101.10	101.90	102.40	103.00	103.60	108.50	1.647
~500	497.00	519.00	526.10	527.10	533.70	563.90	12.079
~900	817.80	850.40	860.90	861.20	871.40	914.20	17.664

Table 3. CPU time statistics of PROBE. The columns report (from left to right) the expected run-time, the minimum, the 1st quartile, the median (2nd quartile), the mean, the 3rd quartile, the maximum and the standard deviation of the observed run-times.

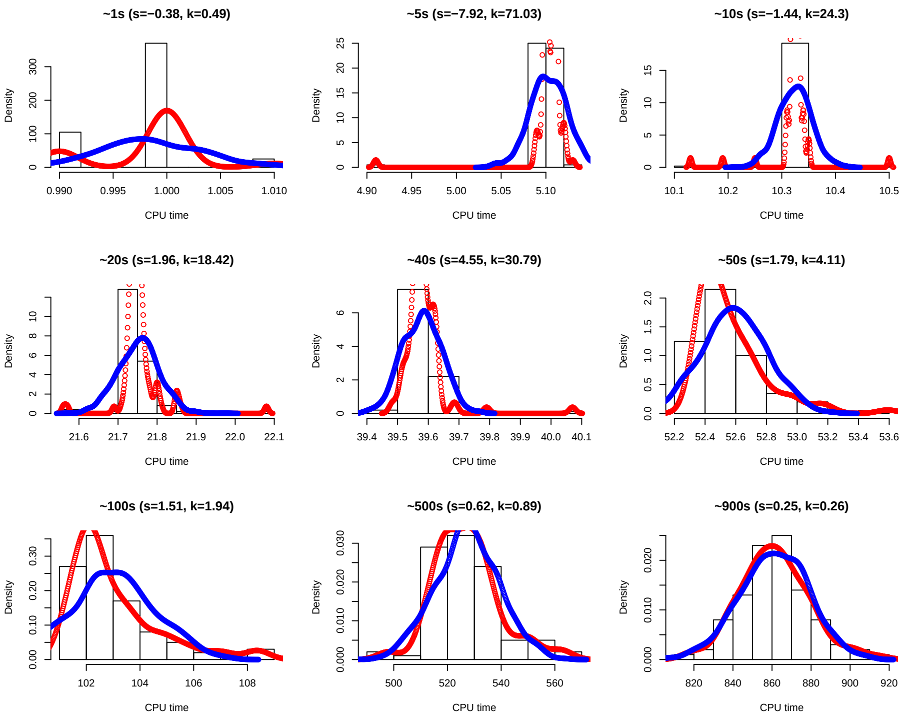

Fig. 4. CPU time frequency distributions of PROBE. Each plot shows an histogram that represents the frequency distribution of the observed values (bins of equal size, total area 1), and two pdf estimates: the red dots are the pdf estimated from observed run-times, while the blue dots represent a normal pdf with the mean and standard deviation computed from the observed run-times. The title of each plot reports the expected run-time, as well as skewness (s) and kurtosis (k) values of the pdf plotted in red (notice that a normal pdf is such that s=k=0).

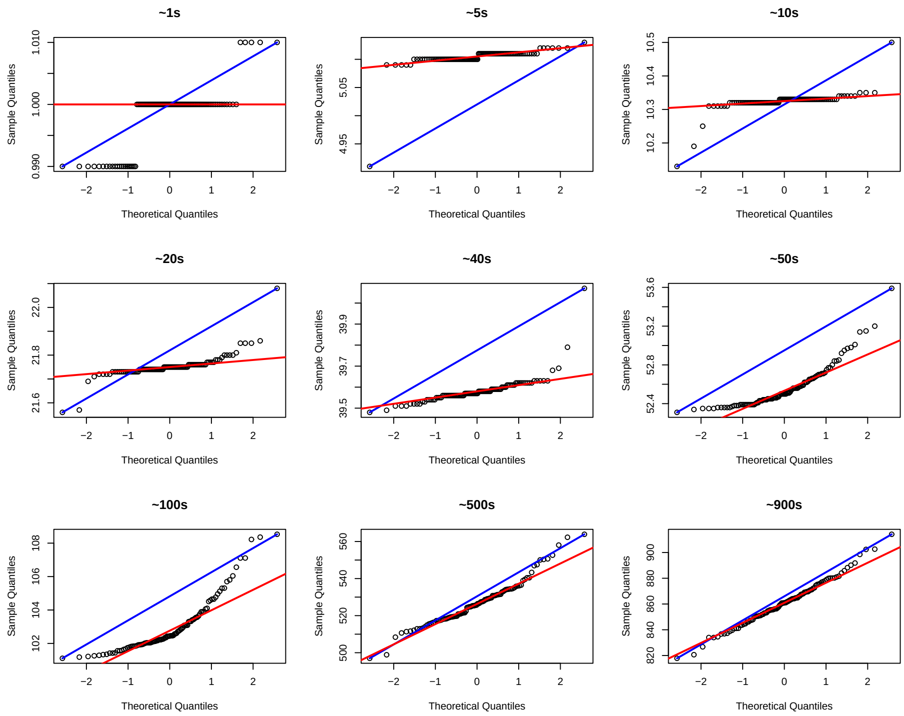

Fig. 5. Normal quantile-quantile plots of the CPU time statistics of PROBE. In the plots, the normal quantiles (Theoretical Quantiles) range on the X axis, while the quantiles of the observed run-times range on the Y axis (Sample Quantiles). The blue line joins the points corresponding to the minimum (bottom-right) and maximum (top-left) observed run-times, while the red line joins the points corresponding to the 1st and 3rd quartile.

Run-time	Mean(μ)	Std. dev.(σ)	Norm	Log-norm	$\mu \pm 3\sigma$	$\hat{\sigma}$	$\mu \pm 3\hat{\sigma}$
~1	0.99	0.005	<0.001	<0.001	100	0.02	100
~5	5.10	0.102	<0.001	<0.001	99	0.021	100
~10	10.32	0.032	<0.001	<0.001	97	0.206	100
~20	21.75	0.051	<0.001	<0.001	97	0.435	100
~40	39.58	0.065	<0.001	<0.001	98	0.792	100
~50	52.57	0.220	<0.001	<0.001	99	1.051	100
~100	103.00	1.647	<0.001	<0.001	97	2.061	100
~500	527.10	12.079	0.005	<0.001	99	10.543	98
~900	817.80	17.664	0.821	<0.001	100	17.224	99

Table 4. Analysis of the CPU time statistics of PROBE. The table reports, going from left to right, the expected run-time, the mean μ and the standard deviation σ computed from the observed run-times; the p-values of the Shapiro-Wilk test for normality and log-normality (only p-values greater than 0.001 are reported); the percentage of observed run-times that fall in the interval $\mu \pm 3\sigma$; the value of $\hat{\sigma}$, i.e., the estimate of σ obtained as described in section 3, and the percentage of observed run-times that fall in the interval $\mu \pm 3\hat{\sigma}$.

B Additional tables and figures of Section 5

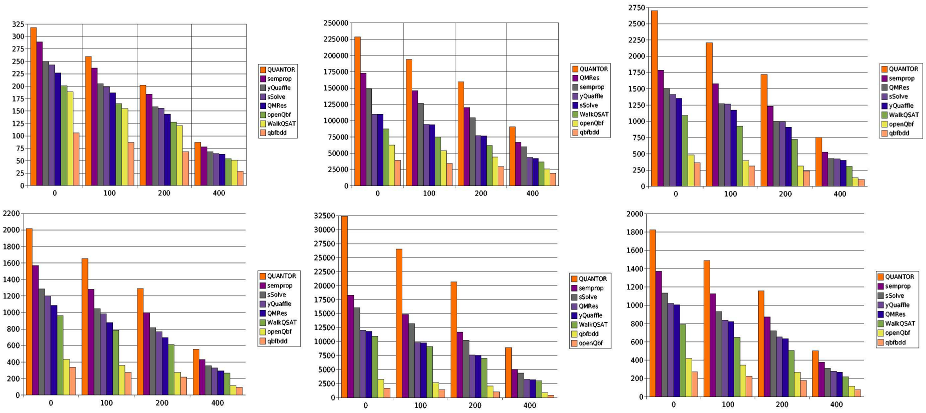

Fig. 6. RDT-stability plots. The fi rst row shows, from left to right, the plots regarding QBF/CASC, SAT and YASMv2 scoring methods, while the second row shows, again from left to right, the plots regarding Borda count, range voting and Schulze's method. Each histogram reports, on the x-axis the number of problems m discarded from the original test set (0, 100, 200 and 400 out of 551) and on the y-axis the score. For each value of the x-axis, eight bars are displayed, corresponding to the scores of the solvers. The legend is sorted according to the ranking computed by the specifi c scoring method, and the bars are also displayed accordingly. This makes easier to identify perturbations of the original ranking, i.e., the leftmost group of bars in each plot corresponding to $m = 0$.

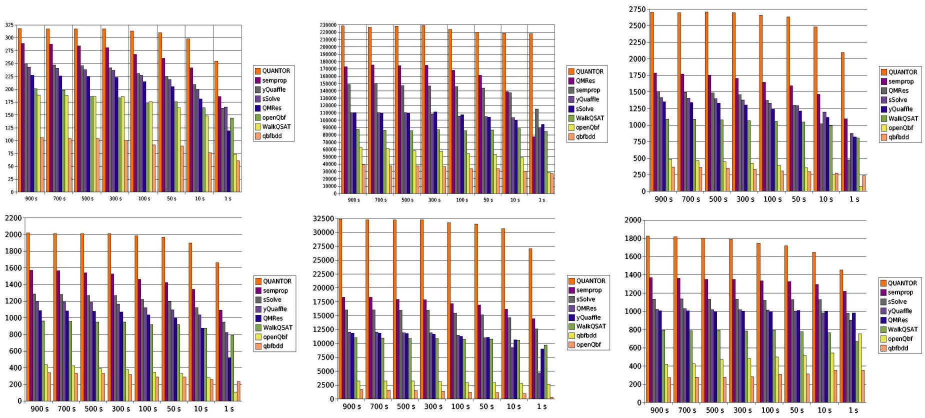

Fig. 7. DTL-stability plots. The histograms are arranged in the same way as Figure 6, except that the x-axis now reports the amount of CPU time seconds used as a time limit when evaluating the scores of the solvers. The leftmost value is $L = 900$, i.e., the original time limit that produces the ranking according to which the legend and the bars are sorted, and then we consider the values $L' = \{700, 500, 300, 100, 50, 10, 1\}$.

	CASC/QBF	SAT	YASM	YASMv2	Borda	r. v.	Schulze
OPENQBF	0.43	0.57	0.36	0.64	0.79	0.79	0.79
QBFBDD	0.43	0.43	0.36	0.64	0.79	0.86	0.79
QMRES	0.64	0.86	0.76	0.79	0.71	0.86	0.71
QUANTOR	1	0.86	0.86	0.86	0.93	0.86	1
SEMPROP	0.93	0.71	0.71	0.79	0.93	0.86	0.93
SSOLVE	0.71	0.57	0.57	0.79	0.86	0.79	0.86
WALKQSAT	0.57	0.57	0.43	0.71	0.64	0.79	0.71
YQUAFFLE	0.71	0.64	0.57	0.71	0.86	0.86	0.86
Mean	0.68	0.65	0.58	0.74	0.81	0.83	0.83

Table 5. Kendall coefficient between the ranking obtained on the original test set and each of the rankings obtained on the solver-biased test sets. Each column is relative to a scoring method, and each row, but the first and the last, is relative to a specific set biased on the named solver. The last row reports the mean value of the coefficients for each scoring method.

	SOTA-distance
CASC	1
QBF	1
SAT	0.71
YASM	0.86
YASM v2	0.79
Borda	0.86
range voting	0.71
Schulze	0.86

Table 6. SOTA-relevance, i.e., Kendall coefficient between the ranking induced by computing the distance of any given solver w.r.t. the SOTA-solver and the ranking obtained by any given scoring method.