Journal on Satisfiability, Boolean Modeling and Computation 2 (2006) 145-164

Report of the Third QBF Solvers Evaluation[*]

Massimo Narizzano mox@dist.unige.it
Luca Pulina luca@star.dist.unige.it
Armando Tacchella tac@dist.unige.it
Laboratory of Systems and Technologies for Automated Rasoning (STAR-Lab)
DIST, University of Genoa, Viale Causa, 13 – 16145 Genoa, Italy

Abstract

This paper reports about the 2005 comparative evaluation of solvers for quantified Boolean formulas (QBFs), the third in a series of non-competitive events established with the aim of assessing the advancements in the field of QBF reasoning and related research. We evaluated thirteen solvers on a test set of more than three thousands QBFs, selected from instances submitted to the evaluation and from those available at www.qbflib.org. In the paper we present the evaluation infrastructure, from the criteria used to assemble the test set to the hardware set up, and we show different views about the results obtained, highlighting the strength of different solvers and the relative hardness of the instances included in the test set.

Keywords: *quantified Boolean formulas, empirical evaluation, automated reasoning tools*

Submitted October 2005; revised February 2006; published March 2006

1. Introduction

The 2005 comparative evaluation of solvers for quantified Boolean formulas (QBFs) is the third in a series of non-competitive events established with the aim of assessing the advancements in the field of QBF reasoning and related research. The non-competitive nature of the evaluation is meant to encourage the developers of QBF reasoning tools and the users of QBF technology to submit their work. Indeed, our evaluation does not award one particular solver, but instead draws a picture of the current state of the art in QBF solvers and applications of QBF reasoning. Running the evaluation enables us to collect data regarding the strength of different solvers and methods, the relative hardness of the QBF instances available, and to shed some light on the open issues for the researchers in the QBF community.

With respect to last year evaluation [1] we witnessed a slight decrease in the number of submitted solvers (from sixteen to thirteen) but we observed a growth in the variety of techniques that are used by the solvers. In particular, while most of the participants in the previous evaluations were complete solvers extending the well-known Davis, Put-

[*] The authors would like to thank the anonymous reviewers for their help in improving the quality of this manuscript, and all the participants to the QBF evaluation for submitting QBF instances and solvers. The work of the authors is partially supported by grants from MIUR and by a grant from Intel Corporation.

nam, Logemann, Loveland procedure (DPLL) [2, 3] for propositional satisfiability (SAT), in this evaluation only seven solvers are DPLL-based. As for the others, one is incomplete (WalkQSAT [4]), one is based on Q-resolution and expansion of quantifiers (quantor [5]), two of them use a core engine based on BDDs: zero-suppressed BDDs in QMRes [6] and standard ROBDDs in qbfbdd [7]. Finally, the solver sKizzo features a combination of the above techniques (including search) and skolemization [8]. The variety of the technologies deployed confirms the vitality of the research on QBF reasoning tools. Regarding applications, two families of QBF instances obtained by encoding, respectively, formal verification and planning problems, have been submitted, for a total of 46 instances. To these we must add three generators, two of them for problems having a circuit-like structure, and one random generator. Finally, a suite of randomly generated Horn/RenHorn QBFs has been submitted.

The evaluation consists of two steps: (*i*) running the solvers on a selection of QBF instances, and (*ii*) analysing the results. Step (*i*) is subject to the stringent requirements of getting meaningful results and completing the evaluation in a reasonable time. In order to fulfill these two requirements, we assembled a test set that minimises the number of instances to be run, while maintaining a reasonable coverage of the initial pool. As we show in Section 3, our resources allowed us to get a complete coverage for some classes of instances, while for others we had to include in the test set only some instances in the space of possible ones. Step (*i*) is completed by running the solvers on a farm of PCs, each solver being restricted to the same amount of time and memory. Step (*ii*) consisted of two phases. In the first phase we considered all the solvers and the instances of the test set to give a rough, but complete, picture of the state of the art in QBF. By analysing the results for discrepancies among the solvers results, i.e., solvers reporting different (un)satisfiability results on the same instance, we were able to isolate solvers that are somehow problematic, and we removed them from the subsequent analysis. The second phase is an in-depth account of the results, where we tried to extract a narrow, but crisp, picture of the current state of the art.

The paper is structured as follows. In Section 2 we briefly describe all the QBF solvers that participated in the evaluation and all the instances that we used to construct the test set. In Section 3 we present the first step of the evaluation, i.e., the choice of the test set and a description of the computing infrastructure where the evaluation ran. In Section 4 we present the results of the evaluation first phase (Section 4.1), including data about discrepancies in the results of the solvers (Section 4.2). In Section 5 we restrict our attention to the solvers that passed the first phase and we present the results of the evaluation, arranged solver-wise in Section 5.1, and instance-wise in Section 5.2. We conclude the paper in Section 6 with a balance about the evaluation, including a discussion of some open problems and challenges for the QBF community at large.

2. Solvers and Instances

Thirteen solvers from ten different authors participated to the evaluation this year. The requirements for the solvers were to be able to read the input instance in a common format (the Q-DIMACS format [9]), and to provide the answer (satisfiable/unsatisfiable) in a given output format. Noticeably, all the solvers complied on the input requirements, which was not the case in previous evaluations [1, 10], while a few solvers required wrapper scripts to adapt the output format (ssolve and semprop) or to load additional applications required to run the solver (openQBF, requiring the JVM). A short description of the solvers submitted to the evaluation follows.

GRL by Andrew G.D. Rowley, is a search-based solver with an innovative learning mechanism described in [11].

openQBF by Gilles Audemard, Daniel Le Berre and Olivier Roussel, is a search-based solver, featuring basic unit propagation and pure literal lookahead, plus a conflict backjumping lookback engine; the heuristic is derived from Böhm and Speckenmeyer's heuristic for SAT.

qbfbdd by Gilles Audemard and Lakhdar Saïs, uses a combination of search and symbolic (BDD-based) techniques (see [7] for more details).

qbflHR by Florian Letombe, is a search-based solver, with a number of features such as trivial-truth, trivial-falsity, Horn and renamable Horn formulas detection.

QchaffLearn by Andrew G.D. Rowley, is a search-based complete QBF solver that uses zchaff as a sub-procedure to determine the (un)satsfiability of the QBF as search proceeds. zchaff is allowed to learn across runs and the QBF solver also implements conflict and solution learning. QChaffLearn uses watched literals and watched clauses [12].

QMRes by Guoqiang Pan and Moshe Y. Vardi, is based on a symbolic implementation of the original DP algorithm, achieved using ZBDDs. The algorithm features multi-resolution, a simple form of unit propagation, and heuristics to choose the variables to eliminate.

quantor by Armin Biere, is based on Q-resolution (to eliminate existential variables) and Shannon expansion (to eliminate universal variables), plus a number of features, such as equivalence reasoning, subsumption checking, pure literal detection, unit propagation, and also a scheduler for the elimination step.

ssolve by Rainer Feldmann and Stefan Schamberger, is a search-based algorithm, featuring trivial truth and a modified version of Rintanen's method of inverting quantifiers. The data structures used are extensions of the data structures of Max Böhm's SAT-solver.

semprop by Reinhold Letz, is a search-based solver written in Bigloo (a dialect of Scheme), featuring dependency directed backtracking and lemma/model caching for false/true subproblems.

WalkQSAT by Ian Gent, Holger Hoos, Andrew G. D. Rowley, and Kevin Smyth, is the first incomplete QBF solver based on stochastic search methods. It is a search-based solver using WalkSAT as a SAT oracle and guidance heuristic.

sKizzo by Marco Benedetti, is a reasoning engine for QBF featuring several techniques, including search, resolution and skolemization (see, e.g., [8]).

yQuaffle by Yinlei Yu and Sharad Malik, is a search-based solver, featuring multiple conflict driven learning, solution based backtracking, and inversion of quantifiers.

The evaluation received 553 instances divided in 11 different families and three random generators, grouped in six different suites (after the name of their authors/submitters) as follows.

Ansotegui (5 families, 38 instances) QBF encodings of evader-pursuer problems on fixed size checkerboards.

Biere A generator for the following problem: given an $\langle n \rangle$-bit-counter with optional reset (r) and enable (e) signals, check whether it is possible to reach the state where all $\langle n \rangle$ bits are set to 1 starting from the initial state where all bits are set to 0.

Pan A generator of barrel-shifter instances, with n control bits (i.e., 2^n input and output bits) with the assertion that for all combinations of input and control bits, there is a combination for the output bits.

Ling (4 families, 8 instances) QBF encodings for FPGA logic synthesis described in [13].

Letombe (2 families, 507 instances) Randomly generated Horn and renamable Horn QBFs (the generator was not part of the submission).

ESTWZ (i.e., Egly, Seidl, Tompits, Woltran and Zolda) A generator of QBF encodings for the evaluation of nested counterfactuals (see [14] for more details).

In order to obtain the evaluation test set, we have also considered about five thousand instances (67 families) from `www.qbflib.org`:

Ayari (5 families, 72 instances) A family of problems related to the formal equivalence checking of partial implementations of circuits (see [15]).

Castellini (3 families, 169 instances) Various QBF-based encodings of the bomb-in-the-toilet planning problem (see [16]).

Gent-Rowley (8 families, 612 instances) Various encodings of the famous "Connect4" game into QBF [17].

Katz (2 families, 20 instances) QBF encodings of Model Checking problems for safety properties.

Lahiri-Seshia (1 family, 3 instances) QBF encodings of convergence testing instances generated in term-level model checking.

Letz (1 family, 14 instances) Formulas proposed in [18] generated according to the pattern $\forall x_1 x_3 \ldots x_{n-1} \exists x_2 x_4 \ldots x_n (c_1 \wedge c_n)$ where $c_1 = x_1 \wedge x_2$, $c_2 = \neg x_1 \wedge \neg x_2$, $c_3 = x_3 \wedge x_4$, $c_4 = \neg x_3 \wedge \neg x_4$, and so on. The instances consists of simple variable-independent subproblems but they should be hard for standard (i.e., without non-chronological backtracking) QBF solvers.

Mneimneh-Sakallah (12 families, 202 instances) QBF encodings of vertex eccentricity calculation in hardware circuits [19].

Narizzano (4 families, 4000 instances) QBF-based encoding of the robot navigation problems presented in [16].

Pan (18 families, 378 instances) Encodings of modal K formulas satisfiability into QBF (see [20]). The original instances have been proposed during the TANCS'98 comparison of theorem provers for modal logics [21].

Rintanen (5 families, 47 instances) Planning, hand-made and random problems, some of which have been presented in [22].

Scholl-Becker (8 families, 64 instances) encode equivalence checking for partial implementations problems (see [23]).

3. Evaluation: test set and infrastructure

As we outlined in Section 1, deciding the test set for the evaluation is complicated by two competing requirements: (i) obtaining meaningful data and (ii) completing the evaluation in reasonable time. As a first step towards this goal, we classified the instances into classes according to their structure: fixed and probabilistic. We say that a class of instances has probabilistic structure when at least one of the parameters characterizing the instances of the class is a random variable following a given probability distribution. On the other hand, if none of the parameters characterizing the instances of a class is a random variable, then the class of instances has fixed structure. Intuitively, in the case of probabilistic structure classes, the parameters that define the members of the class do not pin down specific instances, but instead sets of instances from which a (small) subset of candidates is chosen from a (possibly large) population. Fixed structure classes, on the other hand, exhibit only one representative for each choice of the characteristic parameters: in the limit, fixed structure classes are collection of instances, each one being a unique specimen obtained, e.g., from different encodings of the same real-world application. Fixed structure classes include, for instance, the families generated by iterating some kind of fixed pattern as in Pan/TANCS'98 suite, or those derived from planning domains as Castellini "Bomb in the toilet" and Ansotegui "Evader-pursuer" suites. Probabilistic structure classes include, for instance, the families generated according to Model A [24], or those derived from the Narizzano "Robot on the grid" suite (where a fixed number of obstacles is placed uniformly at random on the grid).[1] Notice that the classification according to structure is essential to avoid compensation effects in the analysis of the results, i.e., some solver could be extremely poor on fixed structure classes, but appear as a good solver overall because of an extremely good performance on probabilistic structure classes. Moreover, since probabilistic structure classes contain (possibly many) samples of each instance, their cardinality is usually large when compared to fixed structure ones, and this could severely bias any analysis that did not differentiate between the two classes.

Table 1 shows the classification of the evaluation test set according to structure, as well as the number of instances in each of the classes and suites included in the set (parenthesised numbers). From Table 1 we see that the evaluation test set is comprised of 551 and 2640 instances coming from fixed and probabilistic structure classes, respectively, for a total of 3191 instances. The test set is divided into three groups. With reference to Table 1, from top to bottom:

- the first group ("Generated", 3 fixed and 2 probabilistic structure suites) contains instances that we generated using software submitted for the evaluation or available on QBFLIB;
- the second group ("Sampled", 5 fixed and 2 probabilistic structure suites) contains instances for which we extracted only some representatives from the original collections: the purpose was to reduce the number of instances while maintaining significant coverage of the original classes;
- the third group ("Covered", 6 fixed structure suites) contains families that could not be easily shrinked without incurring the risk of losing relevant contents: given the relatively small size of these suites, the number of submitted solvers and the available computing resources, this choice was perfectly sustainable.

In particular, the criteria used for assembling the first and second group of instances are detailed in the following.

1. In the previous QBF evaluations [1, 10] an analogous classification used the terminology "random" and "non-random" to identify probabilistic and fixed structure classes respectively.

Table 1. Overview of the QBF evaluation test set.

Selection	Fixed (551)	Probabilistic (2640)
Generated	Biere (24) Gent-Rowley (60) Pan/Shifter (6)	ESTWZ (1080) Tacchella/Model A (1200)
Sampled	Castellini (38) Mneimneh-Sakallah (52) Pan/TANCS'98 (108)	Letombe (240) Narizzano (120)
Covered	Ansotegui (38) Ayari (71) Katz (20) Lahiri-Seshia (3) Letz (14) Ling (8) Rintanen (45) Scholl-Becker (64)	

Biere We generate counters with increasing width, from 2 to 64 bits, with exponential increment (2,4,16,...). For each width, we consider the four versions obtained allowing or suppressing the reset and enable signals.

ESTWZ In the original theory (evaluation of nested counterfactuals over a knowledge base) we consider instances having 4,8 and 16 variables, with 2,4, and 8 clauses-to-variables ratio, and a nesting depth of 2,4, and 8; all the clauses have size 3 and the sample size is 10. The corresponding QBF instance is obtained as a non-prenex QBF, so we also take into account also four different prenexing strategies (described in [14]).

Gent-Rowley The basic Connect4 game is played with 4 stones on a 7x6 board. The parametric version is generated by scaling the board up and down; scaling up yields, 8x7 and 9x8 boards, scaling down yields 6x5 and 5x4 boards. For each board size, we generate other configurations by varying (*i*) the number of stones (from 3 up to the height of the board) and (*ii*) the outcome of the game (either "R"ed wins, "W"hite wins or "D"raw).

Pan/Shifter We generate shifters of increasing widths, from 3 to 8 bits, in step 1 increments; it is important to notice that the file size corresponding to the largest instance is 4MB.

Tacchella/Model A The number of alternations k in the quantifiers goes from 0 (i.e., a SAT formula) to 5 using step 1 increments; the number of variables v is fixed to 20, 40, 80, 160 (the total number of variables is thus $v(k+1)$); for each alternation depth and number of variables, the clause-to-variables ratio varies from 2 to 32 in exponential steps. There are at least 2 existential literals per clause, and clauses have length 5. We consider 10 samples per data point.

Castellini We discard the "G" suite (deterministic) and keep the "A" (nondeterministic, flushing action allowed for the toilets) and the "C" suite (as "A" plus the possibility of clogging). For each of the two suites we run the same combinations of tests used in [16].

Letombe We run a selection of the submitted instances obtained by imposing a coarser sampling of the parameter space.

Mneimneh-Sakallah For each circuit, we run a variable number of instances using exponential increments until the diameter is reached. We include the formula corresponding to the diameter length k (unsatisfiable) and the formula corresponding to $k-1$ (satisfiable). All the other formulas corresponding to $j < k$ are satisfiable.

Narizzano For each value of D (number of obstacles known in advance) we consider plan lengths 2,4 and 8. The sample size is 10.

Pan/TANCS'98 The original Modal K instances are comprised of 9 subseries, each one including 21 satisfiable instances ("n" problems) and 21 unsatisfiable instances ("p" problems) parametrised according to their "size" k. We run 12 QBF instances for each of the 9 subseries, corresponding to 6 "p" and 6 "n" problems, respectively. We consider instances with $k \in \{4, 8, 12, 16, 20, 21\}$.

Notwithstanding the relatively small sample size, notice that the total number of instances in the ESTWZ and Tacchella/Model A suites accounts for more than half of the total number of instances in the test set.

As for the computing infrastructure, the evaluation ran on a farm of 10 identical rack-mount PCs, equipped with 3.2GHz PIV processors, 1GB of RAM and running Debian/GNU Linux (distribution `sarge`). We split the evaluation job evenly across the 10 machines, using `perl` scripts to run subsets of instances on all the 13 solvers on each machine. This methodology has two points in its favour. First, the scripts are extremely lean and simple: one server script, plus as many client scripts as there are machines running, accounting for less than 100 lines of `perl` code. This makes the whole evaluation infrastructure lightweight and easy to debug. Second, by running clusters of instances on the same machine, we are guaranteed that small differences that could exist even in identical hardware, are compensated by the fact that a given instance is evaluated by all the participants on the very same machine. While noise in the order of one second does not matter much when comparing instances to decide their hardness, it can make a big difference when the total runtime on the instance is in in the order of one second or less and we are comparing solvers. Finally, all the solvers were limited to 600 seconds of CPU time and 900MB of memory: in the following, when we say that an instance has been solved, we mean that this happened without exceeding the resource bounds above.

4. Evaluation: first phase results

4.1 Preliminary analysis

In Table 2 and Table 3, we present the raw results of the evaluation concerning, respectively, fixed (551 instances) and probabilistic structure (2640 instances) classes of instances. Each table consists of nine columns that for each solver reports its name (column "Solver"), the total number of instances solved and the cumulative time to solve them (columns "#" and "Time", group "Total"), the number of instances found satisfiable and the time to solve them (columns "#" and "Time", group "Sat"), the number of instances found unsatisfiable and the time to solve them (columns "#" and "Time", group "Unsat"), and, finally, the number of instances uniquely solved and the time to solve them (columns "#" and "Time",

Table 2. Results of the evaluation first phase (fixed structure classes).

Solver	Total		Sat		Unsat		Unique	
	#	Time	#	Time	#	Time	#	Time
sKizzo-v0.5*	346	5349.53	175	3764.85	171	1584.68	–	–
quantor	318	2048.39	151	1099.58	167	948.81	4	755.48
sKizzo-v0.4*	310	4424.50	150	2935.19	160	1489.31	–	–
semprop	289	8333.98	132	2641.80	157	5692.18	1	46.20
QchaffLearn*	268	4741.67	89	2436.57	179	2305.10	26	254.50
yQuaffle	250	7041.47	100	3095.33	150	3946.14	–	–
ssolve	243	6827.56	103	2785.80	140	4041.76	–	–
GRL*	241	7889.24	97	4203.54	144	3685.7	5	2289.66
QMRes	227	5044.58	122	2878.88	105	2165.70	15	1151.64
qbflHR*	208	7781.81	90	3639.60	118	4142.21	1	0.30
openQBF	201	13744.20	78	3076.64	123	10667.50	–	–
WalkQSAT	189	3922.26	96	2988.10	93	934.16	–	–
qbfbdd	106	4868.56	53	3488.26	53	1380.30	–	–

group “Unique”); a “–” (dash) in the last two columns means that the solver did not solve any instance uniquely. Both tables are sorted in descending order, according to the number of instances solved, and, in case of a tie, in ascending order according to the cumulative time taken to solve them. A “*” after the solver’s name denotes the solvers that have been removed from the second phase analysis.

Looking at the results on fixed structure classes in Table 2, we can see that all the solvers, with the only exception of qbfbdd, were able to solve at least 25% of the instances in this category. On the other hand, only four solvers, namely sKizzo (both versions), semprop and quantor, were able to solve more than 50% of the instances. Overall, this indicates that given the current state of the art in QBF reasoning, the performance demand of the fixed structure domains is still exceeding the capabilities of most solvers. The performance of the solvers is also pretty similar: excluding qbfbdd, there are 157 instances (about 28% of the total) separating the strongest solver (sKizzo-v0.5), from the weakest solver (WalkQSAT), and the number of instances solved by the strongest five participants are in the range [346-268], thus spanning only 67 instances (about 14% of the total). Some difference arises when considering the number of instances uniquely solved by a given solver: QchaffLearn, QMRes, GRL, quantor, qbflHR, and semprop are the only solvers able to solve, respectively, 26, 15, 5, 4, 1 and 1 instance(s). There are two important observations regarding this result. First, and most important, the outstanding performance of QchaffLearn is obfuscated by a quite heavy discrepancy record: together with qbflHR, QchaffLearn is the solver that has the highest number of results disagreeing with respect to the majority of the other competitors. Indeed, QchaffLearn has been excluded from the second phase for this reason (see Subsection 4.2 for more details). The second observation is that, excluding QchaffLearn, QMRes stands out as the best solver in terms

Table 3. Results of the evaluation first phase (probabilistic structure classes).

Solver	Total		Sat		Unsat		Unique	
	#	Time	#	Time	#	Time	#	Time
ssolve	1824	26694.60	1060	14577.80	764	12116.80	39	2928.53
semprop	1655	76482.80	927	44603.20	728	31879.60	36	12977.80
WalkQSAT	1475	19709.40	785	10458.10	690	9251.32	35	2433.56
sKizzo-v0.5*	1422	68907.30	702	11083.80	720	57823.40	6	3791.52
QchaffLearn*	1413	32784.30	768	15236.20	645	17548.10	7	2047.92
qbflHR*	1388	26003.30	893	9702.13	495	16301.20	12	11.91
GRL*	1368	27528.20	777	14618.90	591	12909.30	1	366.91
sKizzo-v0.4*	1358	52155.80	656	7365.07	702	44790.70	3	1027.44
yQuaffle	1305	41105.10	747	21233.80	558	19871.30	20	1088.96
openQBF	1272	39971.90	745	23742.90	527	16229	2	1180.99
quantor	821	4588.30	624	3098.37	197	1489.93	–	–
qbfbdd	461	14758.10	333	8737.40	128	6020.67	1	821.62
QMRes	451	24164.10	415	14098.90	36	10065.20	–	–

of instances uniquely solved. This result confirms that the technology on which QMRes is based provides an interesting alternative to the "classic" search-based paradigm.[2.]

Looking at the results on random instances in Table 3, we can see that all the solvers, with the exception of QMRes and qbfbdd, were able to solve at least 25% of the instances in this category; eight solvers were able to solve more than 50% of the instances, but no solver reached the 75% threshold. Overall, this indicates that the choice of parameters for the generation of probablistic classes yielded a performance demand within the capabilities of most solvers, but with a kernel of hard instances that are challenging for the current state of the art. The performance of the solvers is however rather different: even excluding QMRes and qbfbdd, there are 1003 instances (about 38% of the total) separating the strongest solver (ssolve), from the weakest one (quantor), and the number of instances solved by the strongest five participants are spread over 411 instances (about 16% of the total). Considering the number of instances uniquely solved by a given solver, it turns out that ssolve is both the strongest solver *and* the one solving the highest number of instances in this class, but also the performance of semprop, WalkQSAT, yQuaffle and qbflHR stands out. In the case of WalkQSAT, this is probably due to the randomised algorithms on which the solver is based, since they allow WalkQSAT to solve quickly some instances that are out of the reach of complete solvers (e.g., some of the Nested Counterfactuals instances). As for qbflHR, the good performances on some classes of instances (chiefly QHorn and Model A instances) is paramount to obtaining the good overall result. Finally, semprop and yQuaffle do not excel on any single family, but they yield good performances on a wide variety of instances across different probabilistic structure classes.

Table 4. Discrepancies occurred in the results (by solver).

Solver	Majority	Minority
GRL	51	13
openQBF	54	0
qbfbdd	13	0
qbflHR	43	26
QchaffLearn	39	46
QMRes	27	0
quantor	53	0
semprop	75	0
sKizzo-v0.4	48	11
sKizzo-v0.5	50	11
ssolve	80	0
WalkQSAT	73	0
yQuaffle	58	0

4.2 Discrepancies among solver results

As we have anticipated in Section 1, some discrepancies, i.e., at least two solvers reporting different (un)satisfiability results on the same instance, were detected during the analysis of the first phase results. A total of 95 discrepancies were detected, of which 60 regarding probabilistic structure classes and the remaining 35 regarding fixed structure classes. Table 4 reports the data about discrepancies and it is divided into three columns reporting the name of the solver ("Solver"), the number of times its result agrees with the majority of the solvers ("Majority") and the number of times its result disagrees with the majority of the solvers ("Minority"). Considering the results presented so far, we have decided to include in the second phase of the evaluation only those solvers that have a "clean record", i.e., that are never found disagreeing with the majority of solvers on some instance. From Table 4 we can see that the solvers having this property are eight: openQBF, qbfbdd, QMRes, quantor, semprop, ssolve, WalkQSAT, and yQuaffle. On the other hand, GRL, QchaffLearn, qbflHR, and sKizzo (both versions) are excluded on the basis of the previous argument. The data obtained by disregarding the above solvers are free of discrepancies. Clearly, for instances that were solved by just one solver, and for which we do not know the satisfiability status in advance, the possibility that the solver is wrong still exists, but we consider this as unavoidable given the current state of the art. Notice that the same problem exists in the SAT competition when a solver is the only one to report about an instance and the answer is "unsatisfiable".[3.] Finally, we wish to point out that our data, and the policy that we adopted, reflects the state of affairs at the time of the evaluation. Indeed, most of the solvers that we excluded from the second phase have undergone further

2. Indeed, many of the instances uniquely solved by QMRes are in the Pan/TANCS'98 suite, so the performances of QMRes may result from specific optimizations for this kind of problems.
3. This year, the SAT competition hosted a special category reserved to solvers that can emit an unsatisfiability proof, but only two participants (out of more than 50 solvers submitted to the competition) had this capability, confirming the technological difficulty of emitting certificates whenever a full proof of the result is necessary.

Table 5. Results of the evaluation second phase (fixed structure classes).

Category	Solver	Solved		Sat		Unsat		Unique	
		#	Time	#	Time	#	Time	#	Time
Formal Verification (242)	QUANTOR	101	1716.70	53	935.57	48	781.13	12	1578.35
	SEMPROP	83	1853.18	41	371.07	42	1482.11	2	553.47
	SSOLVE	81	2480.64	36	281.12	45	2199.52	2	0.08
	YQUAFFLE	79	1417.83	40	324.75	39	1093.08	–	–
	QMRES	75	2545.13	45	2040.24	30	504.89	18	1288.27
	OPENQBF	68	3339.33	28	934.78	40	2404.55	–	–
	QBFBDD	64	1513.61	29	1139.05	35	374.56	–	–
	WALKQSAT	52	836.18	30	790.40	22	45.78	–	–
Planning (159)	YQUAFFLE	108	2444.83	28	1012.29	80	1432.54	–	–
	SEMPROP	103	1947.95	26	297.13	77	1650.82	1	511.79
	QUANTOR	102	230.70	29	74.43	73	156.27	2	16.23
	SSOLVE	101	883.00	28	302.23	73	580.77	–	–
	OPENQBF	93	8883.30	24	900.29	69	7983.01	–	–
	WALKQSAT	85	236.30	31	86.42	54	149.88	–	–
	QMRES	37	1410.15	10	468.07	27	942.08	–	–
	QBFBDD	26	2704.64	9	1699.03	17	1005.61	–	–
Miscellanea (150)	QUANTOR	115	100.99	69	89.58	46	11.41	2	12.11
	QMRES	115	1089.30	67	370.57	48	718.73	14	142.38
	SEMPROP	103	4532.85	65	1973.6	38	2559.25	3	235.64
	YQUAFFLE	63	3178.81	32	1758.29	31	1420.52	–	–
	SSOLVE	61	3463.92	39	2202.45	22	1261.47	–	–
	WALKQSAT	52	2849.78	35	2111.28	17	738.50	–	–
	OPENQBF	40	1521.53	26	1241.57	14	279.96	–	–
	QBFBDD	16	650.31	15	650.18	1	0.13	–	–

improvements and bug fixes (e.g., new versions of QCHAFFLEARN and SKIZZO have been recently distributed), and new and promising approaches to certify the solvers' output are emerging (e.g., the method described in [25]).

5. Evaluation: second phase results

5.1 Solver-centric view

In Table 5 we report second phase results about fixed structure classes (551 instances), divided into three categories:

Formal Verification 242 instances, including Ayari, Biere, Katz, Mneimneh/Sakallah, and Scholl/Becker instances.

Planning 159 instances, including Castellini, Gent/Rowley, Narizzano, and part of Rintanen instances.

Miscellanea 150 instances, including Letz, Pan (both TANCS'98 and Barrel Shifter) and the remaining Rintanen instances.

Table 5 is arranged analogously to Tables 2 and 3, except an additional column that indicates the category. The solvers are classified independently for each category, and in descending order according to the number of instances solved: in case of ties, the solvers are prioritised according the time taken to solve the instances, smallest time first.

Looking at Table 5, the first observation is that three solvers, namely quantor, semprop and yQuaffle turn out to be the strongest solvers on fixed structure classes: quantor is leader in Formal Verification and Miscellanea categories (in the latter together with QMRes) and it earns a third place in the Planning category by solving only five problems less than the leader yQuaffle; semprop, in spite of its far-from-optimised implementation, turns out to be second best in Formal Verification and Planning categories, and third best on Miscellanea instances; yQuaffle ends up fourth best in Formal Verification, but it is the best solver on Planning instances, and the fourth best on the Miscellanea instances. A second observation regards QMRes which, in spite of its opaque performance on Planning instances, rivals with quantor for the leadership in the Miscellanea category, and is the only solver able to solve 18 instances in the Formal Verification arena and 14 in the Miscellanea category (the highest number of uniquely solved instances in both cases). Overall, the strongest solver in the Formal Verification category (quantor) is able to solve only 42% of the total number of instances, while yQuaffle is able to solve about 68% of the Planning category, and both quantor and QMRes raise to 77% the percentage of solved instances in the Miscellanea category. It is also significant that in the planning category the strongest solver is DPLL-based (yQuaffle), while both in the Formal verification and in the Miscellanea categories the strongest solvers (quantor and QMRes) express alternative paradigms.

Focusing on Formal Verification category, we can see that all the solvers are pretty much in the same capability ballpark: only 49 instances separate the strongest solver (quantor) from the weakest one (WalkQSAT). Considering the three strongest solvers, namely quantor, semprop and ssolve, we can see that they are able to uniquely solve 12, 2 and 2 instances, respectively, so none of them is subsumed by the portfolio constituted by all the other solvers. At the same time, semprop, with 22.33 seconds average solution time, and ssolve, with 30.63 seconds, seem to be slightly less optimised than quantor, which fares around 17 seconds average solution time. Among the other solvers, it is worth noting that WalkQSAT performs quite nicely in terms of average solution time (16 seconds), while QMRes stands out for its ability to uniquely solve 18 instances using a cumulative CPU time that is only twice as big as the time limit (600 seconds) allowed for a single instance.

As for the Planning category, we can see that the performances of the solvers are more diverse than in Formal Verification: 82 instances (about 50% of the total) separate the strongest solver (yQuaffle) from the weakest one in this category (qbfbdd). However, this portion of the test set is relatively easy, as only 1 instance was uniquely solved by semprop (second best) and only 2 instances were uniquely solved by quantor (third best). As noticed before, search-based solvers seem definitely to have an edge over other techniques in this category, as quantor, QMRes and qbfbdd are lagging behind other search-based solvers of comparable strength, such as yQuaffle, ssolve and openQBF. Notice that, although yQuaffle ends up being the strongest solver, the portfolio including all the other solvers is able to solve all the instances that yQuaffle solves.

Finally, considering the Miscellanea category, the first thing to be observed is that most of these instances come from a single source: Pan/TANCS'98 instances (108) and Pan/Shifter (6). In particular, the first group of instances is derived from translations

Table 6. Results of the evaluation second phase (probabilistic structure classes).

Category	Solver	Solved		Sat		Unsat		Unique	
		#	Time	#	Time	#	Time	#	Time
Model A (1200)	ssolve	1126	3461.92	620	1604.62	506	1857.30	57	1782.01
	semprop	1047	5067.83	573	2089.98	474	2977.85	9	202.89
	WalkQSAT	978	6915.97	561	3153.68	417	3762.29	3	142.64
	openQBF	884	15412.53	542	6052.20	342	9360.33	–	–
	yQuaffle	793	12520.38	454	6077.92	339	6442.46	–	–
	quantor	570	3120.84	474	1753.45	96	1367.39	–	–
	QMRes	431	23062.65	403	14049.81	28	9012.84	–	–
	qbfbdd	313	7735.22	199	3016.00	114	4719.22	–	–
Planning (120)	ssolve	88	4568.60	83	2167.56	5	2401.04	–	–
	qbfbdd	87	2416.00	87	2416.00	–	–	2	1482.58
	yQuaffle	85	1041.92	85	1041.92	–	–	–	–
	openQBF	80	12174.73	80	12174.73	–	–	–	–
	semprop	74	10216.09	56	3878.73	18	6337.36	13	5703.50
	quantor	40	79.63	40	79.63	–	–	–	–
	WalkQSAT	1	5.26	1	5.26	–	–	–	–
	QMRes	–	–	–	–	–	–	–	–
QHorn (240)	ssolve	231	542.80	132	537.38	99	5.42	24	132.32
	semprop	141	36124.47	79	22272.33	62	13852.14	–	–
	yQuaffle	137	13158.45	84	3902.50	53	9255.95	–	–
	qbfbdd	57	4605.82	43	3304.37	14	1301.45	1	13.17
	WalkQSAT	51	392.34	–	–	51	392.34	–	–
	openQBF	25	19.47	–	–	25	19.47	–	–
	quantor	15	55.21	–	–	15	55.21	–	–
	QMRes	–	–	–	–	–	–	–	–
Nested Counterfactual (1080)	WalkQSAT	445	12395.86	223	7299.17	222	5096.69	36	2433.60
	semprop	393	25074.39	219	16362.15	174	8712.24	39	11669.82
	ssolve	379	18121.32	225	10268.27	154	7853.05	10	2373.11
	yQuaffle	290	14384.36	124	10211.48	166	4172.88	20	1088.96
	openQBF	283	12365.15	123	5516.00	160	6849.15	2	1180.99
	quantor	196	1332.62	110	1265.29	86	67.33	–	–
	QMRes	20	1101.45	12	49.11	8	1052.34	–	–
	qbfbdd	4	1.03	4	1.03	–	–	–	–

of structured modal K instances [21], and the translation algorithm applied is the same for all the instances, so it is reasonable to assume that the original structure, although obfuscated by the translation, carries over to the QBF instances. In conclusion, the best solvers in this category are probably those that can discover and take advantage of such a hidden structure. Looking at the results it seems that QMRes and quantor are clearly the strongest reasoners in this category. In particular while QMRes is the only one able to solve 14 instances (about 10% of this category), quantor manages to solve the same number of instances of QMRes by using 1 order of magnitude less CPU time, and semprop fares a respectable number of 100 instances solved. Both quantor and semprop are able to solve uniquely 2 and 3 instances, respectively, so none of the three strongest solvers is subsumed by the portfolio including all the solvers.

In Table 6 we report second phase results about random instances (2640 instances), divided into four categories:

Model A 1200 instances, generated according to the guidelines presented in Section 3.

Planning 120 instances, corresponding to the four Robot families in the suite Narizzano.

QHorn 240 instances, corresponding to (a subset of) the quantified Horn and renamable Horn instances submitted to the evaluation by Florian Letombe.

Nested Counterfactuals 1080 instances, generated according to the guidelines presented in Section 3.

Table 6 is arranged analogously to Table 5.

Looking at Table 6, the first observation is that it is difficult to identify a group of solvers that performs well across different categories. For instance, while ssolve turns out to be the best on three categories out of four, namely Model A, Planning, and QHorn, it is only third best on Nested Counterfactuals instances. A second observation is that search-based solvers seem to have an edge over solvers using other kind of techniques. quantor, QMRes and qbfbdd trail the pack on Model A and Nested Counterfactuals instances. On QHorn instances, their performance is far from the strongest solvers in this category, although qbfbdd manages to solve one instance uniquely. The Robot category represents a (partial) exception to this trend, with qbfbdd managing to get an excellent performance in terms of instances solved (only three less than ssolve) and particularly in terms of running time with an average solution time of 27.77 seconds versus 51.90 seconds obtained by ssolve. Notice, however, that in this category the performance of quantor and QMRes is still relatively poor. A third observation relates to the performance of WalkQSAT, which is rather good on Model A and Nested Counterfactuals categories, and quite poor on Planning and QHorn instances. These data seem to indicate that, although all these categories are defined by probabilistic parameters, their peculiar structure is pretty differentiated across the categories.

Focusing on Model A instances, we see that the strongest solver in this category (ssolve) is able to solve 1126 instances, about 94% of the total, while the second and third best solvers (semprop and WalkQSAT) are able to solve 1047 and 978 instances, about 87% and 81% of the total, respectively. ssolve, semprop and WalkQSAT are also the only solvers able to solve 57, 19 and 3 instances, respectively, so none of them is subsumed by the portfolio obtained considering all the second phase solvers. At least in the case of Model A instances it is pretty clear that the best solvers are search-based (WalkQSAT is also incomplete), that the hardness of the test set is well within their capabilities, and that their architecture (particularly the one of ssolve) seems to be best suited to solve this kind of instances.

Regarding Planning instances, we recall that this category coincides with a subset of the four "Robot on the grid" families available on QBFLIB. ssolve is again the strongest solver with 88 (73%) instances solved, while qbfbdd and yQuaffle, solving 87 and 85 instances, are second and third best respectively. The results on this category of problems are rather interesting, both because qbfbdd excellent performances on these instances are unabridged in other categories, and because semprop, far from being the best solver in the category, is able to solve 9 instances uniquely, while all the other solvers (including ssolve and qbfbdd) are subsumed by the portfolio obtained considering all the solvers. The poor performances of WalkQSAT seem to indicate that these problems might have some kind of structural information which qbfbdd is able to exploit. Since quantor solves only 30% of the instances and QMRes is not able to solve a single instance, the kind of structure hidden in the Robot instances is probably entirely different from the one to be found in fixed structure planning classes, where quantor is one of the strongest solvers and qbfbdd is the weakest one.

In the QHorn category, ssolve is clearly the strongest solver, with 231 instances (96.2%) solved and 24 instances uniquely solved. yQuaffle and semprop are second and third best, but they are able to solve only slightly more than 50% of the instances, so their performance is quite far from that of ssolve. It is interesting to notice that, although ssolve does not implement any specific heuristic for detecting Horn clauses, its performance is particularly good on this class of instances. A possible explanation is that, among the general-purpose optimisations used by ssolve, one is effective also for randomly-generated instances built with Horn or renamable Horn clauses.

Finally, the category of Nested Counterfactuals highlights how our choice of generation parameters for this class yields a test set that is far too difficult for current state-of-the-art solvers. WalkQSAT, remarkably the strongest solver in this category, is able to solve 445 instances, only 41% of the total. The five strongest solvers in this category, namely WalkQSAT, semprop, ssolve, yQuaffle and openQBF are all search based and they are able to solve uniquely 36, 39, 10, 20 and 2 instances respectively. These data seem to indicate that search (particularly, stochastically driven one) is at the moment the best strategy to deal with this kind of problems, and that they are currently defying the heuristics featured by state-of-the-art solvers.

5.2 Instance-centric view

In Table 7 we show the classification of the fixed structure classes included in the evaluation test set according to the solvers admitted to the second phase. Table 7 consists of nine columns where for each family of instances we report the name of the family in alphabetical order (column Family), the number of instances included in the family, the number of instances solved, the number of such instances found SAT and the number found UNSAT (group "Overall", columns "#", "N", "S", "U", respectively), the time taken to solve the instances (column "Time"), the number of easy, medium and medium-hard instances (group "Hardness", columns "E", "M", "H"). The number of instances solved and the cumulative time taken for each family is computed considering the "SOTA solver", i.e., all the second phase solvers running in parallel. An instance is thus solved if at least one of the solvers solves it, and the time taken is the best among the times of the solvers that solved the instance. The instances are classified according to their hardness with the following criteria: easy instances are those solved by all the solvers, medium instances are those non-easy instances that could still be solved by at least two solvers, medium-hard instances are those solved by one reasoner only, and hard instances are those that remained unsolved.

According to the data summarised in Table 7, the fixed structure classes consisted of 551 instances, of which 396 have been solved, 188 declared satisfiable and 208 declared unsatisfiable, resulting in 46 easy, 294 medium, 56 medium-hard, and 155 hard instances.

Table 7. Classification of fixed structure classes (second stage data)

Family	Overall				Time	Hardness		
	#	N	S	U		E	M	H
Adder	32	20	12	8	1220.68	3	6	11
Blocks	13	13	4	9	37.51	2	9	2
C432	8	7	3	4	1286.64	0	5	2
C499	8	5	3	2	2.23	0	4	1
C5315	8	4	2	2	0.34	0	3	1
C6288	8	1	1	0	3.99	0	0	1
C880	8	2	2	0	0.75	0	2	0
Chain	12	12	12	0	0.31	2	10	0
comp	8	8	4	4	0.06	1	7	0
Connect4	60	43	0	43	297.08	0	43	0
Counter	24	12	12	0	281.33	4	5	3
DFlipFlop	10	10	0	10	2.73	2	8	0
EP-4x4-log	7	7	7	0	2.51	0	7	0
EP-4x4-std	7	0	0	0	-	0	0	0
EP-6x6-log	8	3	0	3	286.94	0	3	0
EP-6x6-std	8	0	0	0	-	0	0	0
EP-8x8-log	8	4	0	4	537.21	0	3	1
FPGA_FAST	5	5	4	1	0.79	0	5	0
FPGA_SLOW	3	3	1	2	8.96	0	3	0
Impl	10	10	10	0	0.01	6	4	0
jmc_quant	10	4	2	2	61.53	0	1	3
jmc_quant_sqr	10	3	2	1	11.37	0	0	3
k_branch_n	6	2	2	0	46.29	0	1	1
k_branch_p	6	3	0	3	189.46	0	1	2
k_d4_n	6	6	6	0	36.8	0	1	5
k_d4_p	6	6	0	6	2.18	0	6	0
k_dum_n	6	6	6	0	0.13	0	6	0
k_dum_p	6	6	0	6	0.14	0	6	0
k_grz_n	6	6	6	0	14.64	0	4	2
k_grz_p	6	6	0	6	6.63	0	6	0
k_lin_n	6	6	6	0	11.84	0	6	0
k_lin_p	6	6	0	6	0.71	0	6	0
k_path_n	6	6	6	0	0.28	0	6	0
k_path_p	6	6	0	6	0.26	0	6	0
k_ph_n	6	4	4	0	37.34	1	3	0
k_ph_p	6	2	0	2	2.99	0	2	0
k_poly_n	6	6	6	0	0.11	0	6	0
k_poly_p	6	6	0	6	0.06	0	6	0
k_t4p_n	6	6	6	0	85.93	0	1	5
k_t4p_p	6	6	0	6	21.17	0	2	4
Logn	2	2	0	2	0.85	0	2	0
MutexP	7	7	7	0	0.09	2	3	2
Qshifter	6	6	6	0	18.61	1	5	0
s1196	2	0	0	0	-	0	0	0
s1269	5	0	0	0	-	0	0	0
s27	4	4	1	3	0.47	1	3	0
s298	6	2	2	0	120.38	0	1	1
s3330	5	0	0	0	-	0	0	0
s386	4	1	1	0	9.47	0	0	1
s499	6	2	2	0	154.87	0	1	1
s510	7	1	1	0	11.75	0	0	1
s641	4	0	0	0	-	0	0	0
s713	4	1	1	0	209.31	0	0	1
s820	5	1	1	0	54.79	0	0	1
SzymanskiP	12	12	0	12	546.32	0	12	0
term1	8	8	4	4	143.73	0	7	1
Toilet	8	8	5	3	1.27	2	6	0
ToiletA	18	18	8	10	12.15	5	13	0
ToiletC	20	20	11	9	2.81	4	16	0
Tree	14	14	5	9	0.01	5	9	0
uclid	3	0	0	0	-	0	0	0
VonNeumann	10	10	0	10	17.58	0	10	0
z4ml	8	8	4	4	0.01	5	3	0

These results indicate that the selected fixed structure classes are not trivial for current state-of-the-art QBF solvers, since there is a small number of easy instances, and a substantial percentage of medium-to-hard instances. At the same time, the test set is not overwhelming, since most of the non-easy instances could be considered of medium difficulty, i.e., they are solved by at least two solvers. Some of the families submitted for the evaluation turned out to be pretty hard for the solvers: only 12 out of 24 instances in the counter (Biere) instances, and 14 out of 38 in the EP (Evader-Pursuer, Ansotegui) instances, were solved. On the other hand, the Ling suite, consisting of 8 instances in two families was rated as medium difficulty. Some "old" instances that are still pretty hard for current state-of-the-art solvers include the Lahiri-Seshia suite (uclid, 3 instances, all of them hard), the Mneimneh-Sakallah suite (s27, 298, ..., a total of 52 instances of which 40 are hard), and the Katz suite (jmc_quant, 20 instances, of which 13 are hard).

In Table 8 we show the classification of the probabilistic structure classes included in the evaluation test set according to the solvers admitted to the second phase. Table 8 is arranged similarly to Table 7, where the data about Nested Counterfactuals has been summarised in a single entry, QHorn instances are divided into Horn ("Horn") and renamable Horn (renHorn) families, and Robot instances are presented split into four families corresponding to the number of obstacles known in advance. Model A instances are split into 24 classes corresponding to six different values of the alternation depth (from 0 to 5 alternations), each family comprised of instances with different number of variables (20,40,80,160). According to the data summarised in Table 8, this part of the evaluation second phase consisted of 2640 instances, of which 2029 have been solved, 1126 declared satisfiable and

Table 8. Classification of probabilistic structure classes (second phase data)

Family	Overall				Time	Hardness		
	#	N	S	U		E	M	H
CounterFactual	1080	542	272	270	20479.64	2	433	107
Horn	156	153	77	76	130.85	0	142	11
mA-t2-1qbf-5cnf-20var	50	50	40	10	0	50	0	0
mA-t2-1qbf-5cnf-40var	50	50	40	10	0.71	20	30	0
mA-t2-1qbf-5cnf-80var	50	50	40	10	220.3	10	40	0
mA-t2-1qbf-5cnf-160var	50	40	40	0	1.5	1	39	0
mA-t2-2qbf-5cnf-20var	50	50	20	30	0.1	24	26	0
mA-t2-2qbf-5cnf-40var	50	50	16	34	0.45	8	40	2
mA-t2-2qbf-5cnf-80var	50	47	12	35	617.97	0	39	8
mA-t2-2qbf-5cnf-160var	50	39	10	29	368.38	0	30	9
mA-t2-3qbf-5cnf-20var	50	50	30	20	0.06	10	40	0
mA-t2-3qbf-5cnf-40var	50	50	29	21	0.37	0	50	0
mA-t2-3qbf-5cnf-80var	50	48	28	20	694.9	0	46	2
mA-t2-3qbf-5cnf-160var	50	41	21	20	66.12	0	40	1
mA-t2-4qbf-5cnf-20var	50	50	22	28	0.34	1	49	0
mA-t2-4qbf-5cnf-40var	50	50	20	30	83.81	0	47	3
mA-t2-4qbf-5cnf-80var	50	47	20	27	44.49	0	39	8
mA-t2-4qbf-5cnf-160var	50	43	20	23	265.08	0	31	12
mA-t2-5qbf-5cnf-20var	50	50	30	20	0.06	0	50	0
mA-t2-5qbf-5cnf-40var	50	50	30	20	0.14	0	50	0
mA-t2-5qbf-5cnf-80var	50	50	30	20	2.55	0	50	0
mA-t2-5qbf-5cnf-160var	50	48	28	20	61.88	0	42	6
mA-t2-6qbf-5cnf-20var	50	50	29	21	0.14	0	50	0
mA-t2-6qbf-5cnf-40var	50	49	28	21	11.9	0	45	4
mA-t2-6qbf-5cnf-80var	50	49	27	22	1.18	0	40	9
mA-t2-6qbf-5cnf-160var	50	45	24	21	1.82	0	40	5
renHorn	84	83	56	27	133.61	0	69	14
RobotsD2	30	28	25	3	3678.75	0	23	5
RobotsD3	30	27	21	6	1903.32	0	23	4
RobotsD4	30	25	20	5	1445.26	0	22	3
RobotsD5	30	25	21	4	1724.27	0	22	3

903 declared unsatisfiable, resulting in 126 easy, 1687 medium, 216 medium-hard, and 611 hard instances. These results indicate that overall the selected probabilistic classes are within the capabilities of current state-of-the-art QBF solvers, but in some cases they are challenging as much as structured ones. This is the case, e.g., of Nested Counterfactuals, which contributed 538 problems to the count of hard instances.

6. Conclusions

The final balance of the second QBF comparative evaluation can be summarised as follows.

- 13 solvers participated, 12 complete and 1 incomplete: 8 search-based algorithms, and 5 based on other techniques including Q-resolution, symbolic DP, skolemization, and Shannon expansion.
- 553 instances and three generators were submitted.
- State-of-the-art solvers, both for fixed and probabilistic classes, have been identified; also, a total of 693 challenging instances that could not be solved by any of the

participants admitted to the second phase have been identified to set the reference point for future developments in the field.

The evaluation also evidenced some critical points.

- QBF encodings of real-world applications (e.g., Ayari's hardware verification problems, Sakallah's vertex eccentricity problems, etc.) are still contributing to the pool of challenging instances, confirming analogous results obtained in the previous evaluations.

- With the notable exception of sKizzo and qbfbdd, most of the solvers submitted to the third evaluation have been submitted also to the second one, and some of them have been around since the first edition; among these, ssolve and semprop are essentially the same solvers that were submitted three years ago, but they still manage to maintain a strong standing in some categories (including applications of QBF reasoning).

- The question of how to check the answer of the QBF solvers in an effective way is still mainly unanswered. Some progress has been made on this front, e.g., sKizzo includes a certifier for the results it produces, and in [25] the authors describe a method to generate certifiable results for yQuaffle, but the question of what is a general and reasonably efficient certificate of (un)satisfiability for QBF is still open.

The last point is not only an issue for the QBF evaluation, but also for the implementation of QBF solvers: indeed, only eight out of thirteen solvers passed to the second phase, meaning that developers would benefit a lot from a feasible method that is not linked to a specific solver for checking the results of their solvers before the evaluation begins.

The analyses herewith presented are just a fraction of the possible ones about the evaluation data. For further information, tables, instances and downloads, please visit QBFLIB at www.qbflib.org. Here you will be able to access the evaluation data, from the runtime of specific solvers on specific instances, to cumulative tables similar to those presented in this paper.

References

[1] 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. to appear.

[2] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, **7**(3):201–215, 1960.

[3] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, **5**(7):394–397, 1962.

[4] A. G. D. Rowley I. P. Gent, H. H. Hoos and K. Smyth. Using Stochastic Local Search to Solve Quantified Boolean Formulae. In *9th Conference on Principles and Practice*

of Constraint Programming (CP 2003), volume **2833** of *Lecture Notes in Computer Science*. Springer Verlag, 2003.

[5] A. Biere. Resolve and Expand. In *Seventh Intl. Conference on Theory and Applications of Satisfiability Testing*, 2004. Extended Abstract.

[6] Guoqiang Pan and Moshe Y. Vardi. Symbolic Decision Procedures for QBF. In *10th Conference on Principles and Practice of Constraint Programming (CP 2004)*, 2004.

[7] G. Audemard and Lakhdar Saïs. A Symbolic Search Based Approach for Quantified Boolean Formulas. In *Eight International Conference on Theory and Applications of Satisfiability Testing (SAT 2005)*, volume **3569** of *Lecture Notes in Computer Science*. Springer Verlag, 2005.

[8] 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.

[9] E. Giunchiglia, M. Narizzano, and A. Tacchella. Quantified Boolean Formulas satisfiability library (QBFLIB), 2001. www.qbflib.org.

[10] 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.

[11] Andrew G D Rowley Ian P Gent. Solution learning and solution directed backjumping revisited. Technical Report APES-80-2004, APES Research Group, February 2004.

[12] I.P. Gent, E. Giunchiglia, M. Narizzano, A. Rowley, and A. Tacchella. Watched Data Structures for 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.

[13] A. Ling and S. D. Brown D. P. Singh. FPGA Synthesis Using Quantified Boolean Satisfiability. In *Eight International Conference on Theory and Applications of Satisfiability Testing (SAT 2005)*, volume **3569** of *Lecture Notes in Computer Science*. Springer Verlag, 2005.

[14] U. Egly, M. Seidl, H. Tompits, S. Woltran, and M. Zolda. Comparing Different Prenexing Strategies for Quantified Boolean Formulas. In *Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, volume **2919** of *Lecture Notes in Computer Science*. Springer Verlag, 2003.

[15] Abdelwaheb Ayari and David Basin. Bounded model construction for monadic second-order logics. In *12th International Conference on Computer-Aided Verification (CAV'00)*, number 1855 in LNCS, pages 99–113. Springer-Verlag, 2000.

[16] C. Castellini and E. Giunchiglia and A. Tacchella. SAT-based planning in complex domains: Concurrency, constraints and nondeterminism. *Artificial Intelligence*, **147**:85–117, 2003.

[17] I.P. Gent and A.G.D. Rowley. Encoding Connect 4 using Quantified Boolean Formulae. Technical Report APES-68-2003, APES Research Group, July 2003.

[18] R. Letz. Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas. In *Proceedings of Tableaux 2002*, LNAI 2381, pages 160–175. Springer, 2002.

[19] M. Mneimneh and K. Sakallah. Computing Vertex Eccentricity in Exponentially Large Graphs: QBF Formulation and Solution. In *Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, volume **2919** of *Lecture Notes in Computer Science*. Springer Verlag, 2003.

[20] Guoqiang Pan and Moshe Y. Vardi. Optimizing a BDD-based modal solver. In *Proceedings of the 19th International Conference on Automated Deduction*, 2003.

[21] P. Balsiger, A. Heuerding, and S. Schwendimann. A Benchmark Method for the Propositional Modal Logics K, KT, S4. *Journal of Automated Reasoning*, **24**(3):297–317, 2000.

[22] J. T. Rintanen. Improvements to the Evaluation of Quantified Boolean Formulae. In *Proc. of IJCAI*, pages 1192–1197, 1999.

[23] C. Scholl and B. Becker. Checking equivalence for partial implementations. In *38th Design Automation Conference (DAC'01)*, 2001.

[24] Ian Gent and Toby Walsh. Beyond NP: the QSAT phase transition. In *Proc. of AAAI*, pages 648–653, 1999.

[25] Y. Yu and S. Malik. Verifying the Correctness of Quantified Boolean Formula(QBF) Solvers: Theory and Practice. In *ASP-DAC*, 2005.