Challenges in the QBF arena: the SAT'03 evaluation of QBF solvers — DRAFT —

Daniel Le Berre[1], Laurent Simon[2], and Armando Tacchella[3]

[1] CRIL, Université d'Artois,
Rue Jean Souvraz SP 18 – F 62307 Lens Cedex, France
leberre@cril.univ-artois.fr
[2] LRI, Université Paris-Sud
Bâtiment 490, U.M.R. CNRS 8623 – 91405 Orsay Cedex, France
simon@lri.fr
[3] DIST, Università di Genova,
Viale Causa, 13 – 16145 Genova, Italy
tac@mrg.dist.unige.it

Abstract. The implementation of effective reasoning tools for deciding the satisfiability of Quantified Boolean Formulas (QBFs) is an important issue in several research fields such as Formal Verification, Planning, and Reasoning about Knowledge. Several QBF solvers have been implemented in the last few years, most of them extending the well-known Davis, Putnam, Logemann, Loveland procedure (DPLL) for propositional satisfiability (SAT). At the same time, a consistent breed of QBF benchmarks emerged, both in the form of statistical models for the generation of random formulas, and in the form of real-world instances. In this paper we report about the – first ever – evaluation of QBF solvers that was run as a joint event to SAT'03 Conference on Theory and Applications of Satisfiability Testing. Owing to the relative youngness of QBF tools and applications, we decided to run the comparison on a non-competitive basis, using the same technology that powered SAT'02 and SAT'03 competitions of SAT solvers. Running the evaluation enabled us to collect all sorts of data regarding the relative strength of different solvers and methods, the quality of the benchmarks, and to understand some of the current challenges for researchers involved in the QBF arena.

1 Introduction

The implementation of effective reasoning tools for deciding the satisfiability of Quantified Boolean Formulas (QBFs) is an important issue in several research fields such as Formal Verification [1, 2], Planning [3, 4], and Reasoning about Knowledge [5]. Several QBF solvers have been implemented in the last few years, most of them extending the well-known Davis, Putnam, Logemann, Loveland procedure (DPLL) [6, 7] for propositional satisfiability (SAT). Starting from the seminal papers of Kleine-Büning, Karpinski and Flögel about the system QKN and, later on, of Cadoli, Giovanardi and Schaerf [8] about the system EVALUATE, we have witnessed the development of several complete QBF solvers, such as DECIDE [3] (and its new version QSAT [9]), OPENQBF, QBFL, QSOLVE [10] (and its new version SSOLVE), QUAFFLE [11], QUBE [12], SEMPROP [13]. WALKQSAT [14], the first QBF solver leveraging incomplete stochastic search, has also been recently introduced.

Accompanying the evolution of QBF solvers, a consistent breed of QBF benchmarks emerged, both in the form of statistical models for the generation of random formulas [15, 16], and in the form of real-world instances [3, 2, 1, 4, 5]. In particular, Cadoli et al. in [15] proposed an extension of the fixed clause length generation method for SAT [17] to QBF, which was later improved by Gent and Walsh in [16] to eliminate trivial instances. The generation model named "A" in [16] has been used in other papers (e.g., [10, 18]) to experimentally evaluate heuristics and optimization techniques. Since the problem of deciding the satisfiability of a QBF is paradigmatic for the class of PSPACE-complete problems, there is a consistent number of real-world reasoning tasks that can be recast into the task of solving a QBF. Examples come from the planning community (see, e.g., [3, 4]), where planning domains described with complex action languages have been converted and successfully solved using QBF-based reasoning, and from the formal verification community (see, e.g., [2, 1]), where QBF-based encodings started to show up as an alternative to SAT-based encodings for hardware verification problems. Even more recently Vardi and Pan showed how it is possible to convert problems expressed in the modal logic K into equi-satisfiable instances expressed in QBF in [5]. As reported in [2, 1] and in [5], QBF solvers are still lagging behind special purpose SAT and modal-K solvers, respectively. Therefore the effective application of QBF tools on real-world reasoning tasks is still a challenging issue.

In this paper we report about the – first ever – evaluation of QBF solvers that was run as a joint event to SAT'03 Conference on Theory and Applications of Satisfiability Testing [19]. Owing to the relative youngness of QBF tools and applications, we decided to run the comparison on a non-competitive basis, using SatEx [20], the same technology that powered SAT'02 and SAT'03 competitions of SAT solvers. The evaluation was not intended to award one particular solver, but instead to draw a picture of the current state of the art in QBF solvers and benchmarks. Indeed, running the evaluation enabled us to collect all sorts of data regarding the relative strength of different solvers and methods, the quality of the benchmarks, and to understand some of the current challenges for researchers involved in the QBF arena.

The paper is structured as follows. In Section 2 we describe all the QBF solvers that participated in the evaluation (Sub. 2.1) and the benchmarks that we used to compare the solvers (Sub. 2.2). In Section 3 we present the results of the evaluation arranged by solver (Sub. 3.2), and by benchmark category (Sub. 3.1); in Sub. 3.3 we complete the picture by highlighting the relative merits of different solvers, as well as providing some details about the difficulty of the currently available benchmarks. Section 4 proposes some challenges for the QBF community at large. We conclude the paper in Section 5 with a balance about the evaluation.

2 State of the art in QBF

2.1 QBF solvers

11 solvers participated to the evaluation. The requirements where similar to the ones of the SAT competition: reading the problem in a common format (the Q-DIMACS format [21]) and providing the answer in a given output format. Among the 11 solvers, many did not directly input problems in the Q-DIMACS format. Preprocessing tools to convert the benchmarks from the Q-DIMACS format to each solver format were available, but it was necessary to convert each benchmark into 4 different formats in order to run the evaluation. Furthermore, many of the solvers did not respect the output

requirements, so we needed to either go through the source code to adjust it, or to modify SatEx scripts.

A short description of the solvers submitted to the evaluation follows.

- openQBF by Gilles Audemard, Daniel Le Berre and Olivier Roussel. The solver is a simple extension to QBF of the standard DPLL algorithm. It uses a branching rule based on the quantifier order in the prefix, performs unit propagation and pure literal propagation as described in [15]. The solver, written in Java, is based on the OpenSAT framework [22] and it uses the watched literals data structure [23].
- qbfl by Florian Letombe. A simple and evolutionary C solver that uses Limmat (version 1.3) SAT-solver as a sub-procedure. It is simple, because it is a standard DPLL that implements Cadoli, et al. [15] trivial falsity and truth. It is evolutionary, because frequency for testing triviality changes with the success rate of the tests. It choses variables based on Böhm's and Speckenmeyer's lexicographic order [24]. Unit and pure literal propagation for QBF [15] are implemented too.
- qsat by Jussi Rintanen [3, 9]. An extension of the DPLL procedure featuring a lookahead heuristic with failed literal rule, sampling, partial unfolding and quantifier inversion (see [9] for a description of these techniques).
- ssolve by Rainer Feldmann and Stefan Schamberger [10]. The ssolve algorithm is a DPLL-based search algorithm. It includes the test for trivial truth of Cadoli et al. [15] and a modified version of Rintanen's method of inverting quantifiers [3]. The data structures used are extensions of the data structures of Max Böhm's SAT-solver [37]. The algorithm is the basis of a parallel QBF-solver, which allows for the efficient use of more than a hundred processors.
- quaffle by Lintao Zhang and Sharad Malik [11, 25]. This solver is the extension to QBF of the chaff [23] architecture for SAT. quaffle inherits from chaff efficient data structures for unit clause propagation, and features non-chronological backtracking based on conflict/solution learning [11].
- quantor by Armin Biere. This solver features elimination of innermost universal quantifiers by copying. It is a very preliminary version, which was assembled in less than 2 person weeks. A lot of optimizations are reported to be missing [26].
- QuBE-Bj by Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella [27]. The basic architecture of QuBE is a standard counter-based structure, featuring unit propagation, pure literal propagation, and trivial truth (in the spirit of [15]); the heuristic is and adaptation to QBF of the Böhm's and Speckenmeyer's heuristic for SAT. QuBE-Bj adds to this conflict and solution backjumping, while
- QuBE-Rel by Massimo Narizzano, Armando Tacchella and Enrico Giunchiglia [28] also features good (solution) and no-good (conflict) learning, plus unit clause propagation on universal literals (see [28] for details).
- semprop from Reinhold Letz [29]. Semantic tree (DPLL) with dependency directed backtracking and lemma/model caching for false/true subproblems.
- watchedCSBj Andrew G. D. Rowley and Ian Gent features lazy data structures similar to those of quaffle (see [30]), constraint and solution backjumping in the spirit of QuBE-Bj.
- walkQsat by Andrew G. D. Rowley, Ian Gent, Holger Hoos and Kevin Smyth [14] is the first incomplete QBF solver based on stochastic search methods. It is a version of watchedCSBj with Walksat [31] as a SAT oracle and guidance heuristic. walkQsat is reported to be "an inherently incomplete QBF solver, which still solves many unsatisfiable instances as well as satisfiable ones" [14].

Series		Satisfiability		Formulas Data				
name	count	T%	F%	variables	clauses	sets	exist	forall
k_branch_n	21	100%	0%	5679.05	28927.43	27.00	5614.67	64.38
k_branch_p	21	0%	100%	5679.05	28928.43	27.00	5614.67	64.38
k_d4_n	21	100%	0%	1000.90	3542.38	30.90	952.00	48.90
k_d4_p	21	0%	0%	694.62	2050.33	30.90	656.71	37.90
k_dum_n	21	100%	0%	575.86	1479.67	32.90	546.05	29.81
k_dum_p	21	0%	100%	486.52	1266.81	26.14	462.38	24.14
k_grz_n	21	100%	0%	539.76	1973.00	17.00	520.57	19.19
k_grz_p	21	0%	100%	520.10	1908.38	17.00	500.05	20.05
k_lin_n	21	100%	0%	2276.19	27013.90	8.81	2261.90	14.29
k_lin_p	21	0%	100%	586.24	3220.10	7.00	576.00	10.24
k_path_n	21	100%	0%	802.29	2229.76	27.00	765.29	37.00
k_path_p	21	0%	100%	736.52	2047.67	25.00	702.57	33.95
k_ph_n	21	100%	0%	3211.67	70922.62	4.86	3203.38	8.29
k_ph_p	21	0%	100%	3517.00	79384.52	4.90	3508.62	8.38
k_poly_n	21	100%	0%	911.14	2055.95	72.05	842.10	69.05
k_poly_p	21	0%	100%	909.86	2053.05	71.95	840.90	68.95
k_t4p_n	21	100%	0%	1603.76	5029.76	55.00	1528.81	74.95
k_t4p_p	21	0%	100%	908.81	2815.00	35.00	863.81	45.00
C432	8	?	?	594.50	1525.00	6.50	588.00	6.50
C499	8	?	?	881.50	2528.50	7.00	875.00	6.50
C5315	8	?	?	5509.00	14818.25	10.00	5459.50	49.00
C6288	8	?	?	4702.25	13798.75	6.00	4651.25	50.50
C880	8	?	?	1009.00	2571.50	7.50	994.50	14.00
comp	8	50%	50%	299.75	815.25	4.00	297.25	2.50
term1	8	?	?	1090.75	3760.25	8.00	1084.75	5.75
z4ml	8	50%	50%	63.75	193.00	3.50	62.25	1.50

Table 1. Data of submitted benchmarks.

2.2 QBF benchmarks

In this Section we present the benchmarks used in the evaluation. A total of 1720 benchmarks into 134 series have been run during the evaluation. Of these, 856 are generated according to some probabilistic model, and 864 are non-random (i.e., real-world and hand-made) instances. 442 instances have been submitted for the evaluation, and 1278 have been selected from the instances available on QBFLIB [21]. At the time of this report, all the instances used for the evaluation (and more) are available on QBFLIB.

In more details, the submitted benchmarks are:

Scholl/Becker 64 benchmarks in 8 series, encode equivalence checking for partial implementations problems (see [1]).

Guoqiang Pan 378 benchmarks in 18 series, encode the satisfiability of modal K formulas into QBF (see [5]). The originating benchmarks have been proposed during the TANCS'98 comparison of theorem provers for modal logics [33].

The benchmarks that have been selected from QBFLIB are:

Ayari 72 benchmarks in 8 series. A family of problems related to the formal equivalence checking of partial implementations of circuits (see [2]).

Castellini 169 benchmarks in 5 series. Various QBF-based encodings of the bomb-in-the-toilet planning problem (see [4]).

Letz 14 benchmarks. A family of formulas proposed in [29] 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.

Narizzano/Random 842 benchmarks in 82 series. QBFs randomly generated with model A proposed by Gent and Walsh in [16].

Series		Satisfiability		Formulas Data				
name	count	T%	F%	variables	clauses	sets	exist	forall
Adder-s1	8	100%	0%	2775.00	3742.50	4.00	1216.50	667.50
Adder-s2	8	100%	0%	9071.00	9927.50	6.00	7623.50	556.50
Adder-u1	8	0%	100%	2791.00	3750.50	3.00	1556.50	343.50
Adder-u2	8	0%	100%	9071.00	9919.50	7.00	7725.50	454.50
DFlipFlop	10	0%	100%	62919.50	82655.30	3.00	62058.50	37.50
MutexP	7	100%	0%	8890.43	4172.71	2.00	3078.86	145.14
SzymanskiP	12	100%	0%	82461.75	90270.75	3.00	72526.08	1786.67
VonNeumann	11	0%	100%	441489.00	643479.00	3.00	438327.00	240.00
ToiletA-s	21	100%	0%	304.00	4341.81	3.00	297.43	6.57
ToiletA	56	0%	100%	219.25	5246.98	3.00	211.61	7.64
ToiletC-s	29	100%	0%	406.03	2997.93	3.00	402.97	3.07
ToiletC	56	0%	100%	214.79	986.48	3.00	211.61	3.18
ToiletG	7	100%	0%	49.57	204.29	3.00	46.43	3.14
Tree	14	35%	65%	52.86	34.14	39.29	26.43	26.43
Robots_D2	30	70%	0%	6422.00	21359.00	2.00	5172.50	27.50
Robots_D3	30	57%	30%	6422.00	21360.00	2.00	5172.50	27.50
Robots_D4	30	47%	53%	6422.00	21361.00	2.00	5172.50	27.50
Robots_D5	30	47%	53%	6422.00	21362.00	2.00	5172.50	27.50
Blocks-s	4	100%	0%	435.75	5098.00	3.00	431.00	4.75
Blocks	9	0%	100%	507.11	7570.44	3.00	501.78	5.33
Chain	12	100%	0%	1997.50	11173.75	3.00	1980.00	17.50
Impl	10	100%	0%	46.00	90.00	23.00	35.00	11.00
Logn	4	0%	100%	1318.00	61599.00	2.00	1317.50	0.50
R3cnf-s	13	100%	0%	150.00	383.08	5.15	135.00	15.00
R3cnf	7	0%	100%	150.00	381.43	4.71	135.00	15.00
Toilet-s	5	100%	0%	754.80	3434.60	3.00	751.80	3.00
Toilet	3	0%	100%	240.33	869.00	3.00	238.00	2.33

Table 2. Data of benchmarks from QBFLIB.

Narizzano/Robot 120 benchmarks in 4 series. QBF-based encoding of the "robot-on-the-grid" planning problem presented in [4].

Rintanen 67 benchmarks (44 true, 23 false) in 9 series consisting of planning, hand-made and random problems, some of which have been presented in [3].

To run the submitted solvers on the above benchmarks we used a farm of eight identical PCs equipped with a Pentium IV 2.4 Ghz processor, and 1024MB of RAM (DDR 333Mhz), running Linux RedHat 7.2 (kernel version 2.4.7-10). For practical reasons, a timeout of 15 CPU minutes has been placed on the run time of the solvers on each instance.

3 Results of the evaluation

3.1 Benchmark-centric view

In Tables 2 and 3 we summarize the data about the benchmarks for the submitted instances (Table 2) and for the the instances from QBFLIB (Table 3). In the tables each row represents different families of instances (series) for which we report the number of benchmarks in the family (count), the percentage of true (T%) and false (F%) instances, the average number of variables (variables), clauses (clauses) and alternations in the prefix (sets); we also report the average number of existentially quantified variables (exists) and of universally quantified variables (forall). Notice that:

- when the sum of T% and F% is less than 100%, it means that the status of the problems cannot be known in advance, and not all the benchmarks could be solved, so the numbers shown are the best current estimate of the satisfiability status of the formulas;

solver	Overall		Random			Non-random		
	#series	#benchs	#series	#benchs	Remark	#series	#benchs	Remark
openqbf	110	725	65	404		45	321	
qbfl	118	709	75	415		43	294	
qsat	130	1152	82	701		48	451	SOTAC(22)
quaffle	99	812	60	389		39	423	SOTAC(9)
quantor	64	456	19	115		45	341	
qubebj	131	1383	82	842	SOTAC(4)	49	541	SOTAC(8)
quberel	133	1324	82	803		51	521	SOTAC(12)
semprop	133	1389	82	810		51	579	SOTAC(58)
ssolve	131	1293	82	840	SOTAC(5)	49	453	
walkqsat	117	823	80	600		37	223	
watchedcbsj	125	1004	80	602		45	402	SOTAC(2)

Table 3. Overall results of the evaluation.

– for the sake of compactness, we do not report about the Random benchmarks generated according to model A in detail. Indeed, these are a selection of fixed clause length QBFs (clause length is 5), with 25, 50, 100 and 150 variables per quantifier set, having 2,3,4,5 quantifier sets and different number of clauses. The number of clauses is empirically chosen to cover the 100%-0% satisfiability curve.

3.2 Solver-centric view

Table 4 summarizes the results of the evaluation for all the solvers on all the benchmarks. In the Table we report for each solver the total number of series solved (Overall - #series), the total number of benchmarks solved (Overall - #benchs), and then we present the same data separating random (columns 4-6) and non-random (columns 7-9) instances. A series is considered *solved* iff at least one of its benchmarks is solved, and a solver is considered a *state-of-the-art contributor* (SOTAC) iff it was the only one able to solve a given problem (as in [34]). The columns "Remark" indicate whether a given solver is SOTAC and, if so, how many problems it uniquely conquered. Looking at Table 4 we immediately see that two QBF solvers, SEMPROP and QUBE-REL, were able to solve all but one series (133 out of 134 series) and that SEMPROP, was able to solve the highest number of benchmarks with 1389 instances solved out of 1720. Also, with the exception of QUANTOR, most solvers did quite well on the random benchmarks, while non-random ones proved more challenging for all the solvers. Notice that all series where solved by at least one solver. Interestingly, all the solvers whose authors have published consolidated results turned out to be SOTAC. In the following, we discuss the results of the evaluation considering the results on random and non-random instances separately.

Random benchmarks Figure 1 presents a pictorial view of the evaluation results considering random instances only. The plot in Figure 1 is obtained as follows: the x-axis represents the number of benchmarks solved and the y-axis the time taken to solve a particular instance; the curve representing the performances of the solver is obtained by sorting its CPU times in ascending order. Therefore, the curve represents the distribution of run times for the given solver: the first point is the 0% percentile, the intermediate point is the 50% percentile (median), and so on. The distributions have all the same qualitative behavior, where a solver can easily conquer a few problems and then inevitably reaches its "level of incompetence" (similar phenomena are noticed in [34]).

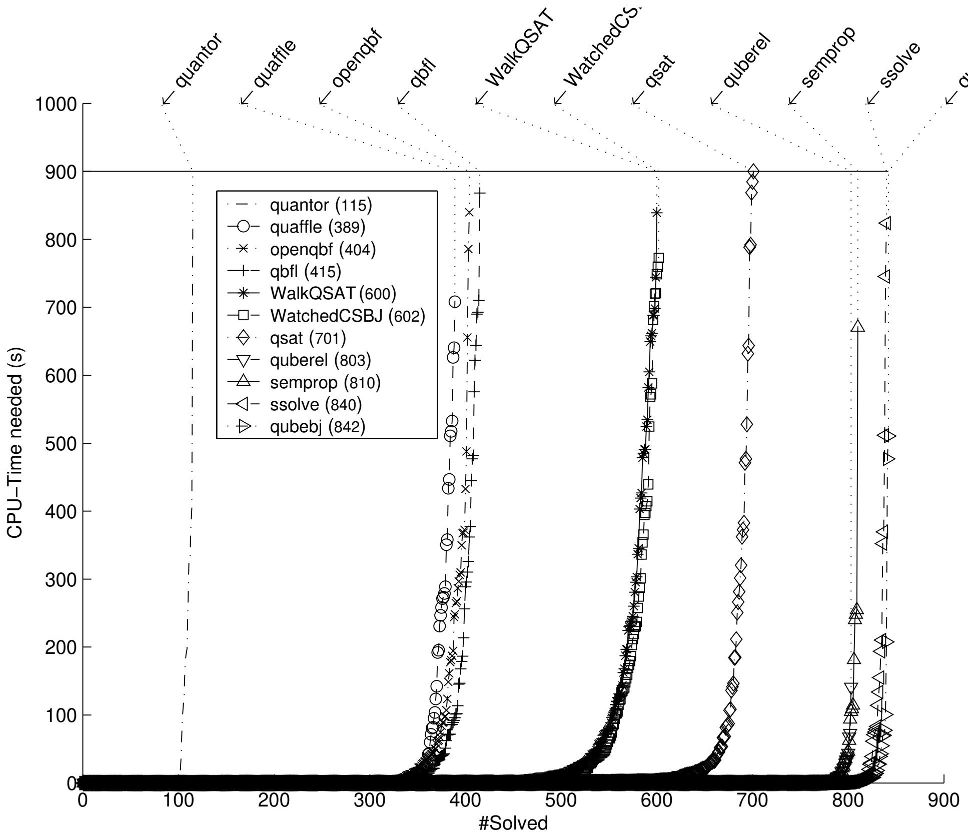

Fig. 1. Number of instances solved vs. CPU time for random benchmarks

Figure 1 shows clearly that solvers are clustered in groups of similar ability as far as solving random benchmarks is concerned. Starting to analyze the plot in Figure 1 from left (worse) to right (best), we see that QUANTOR is by far the worst solver on this category (only 115 benchmarks solved), so it is likely that the approach used in QUANTOR is not adapted to random problems; then we see a cluster made of QUAFFLE, QBFL and OPENQBF (around 400 benchmarks solved). Interestingly, such a sophisticated solver as QUAFFLE is lagging behind very simple ones such as OPENQBF and QBFL. It is likely that the explanation is twofold: the heuristic used by QUAFFLE (VSIDS) is probably not adapted to random problems, and, perhaps more importantly, the pure literal rule is missing from QUAFFLE. Since both QBFL and OPENQBF perform pure literal detection, we conclude that this rule seems quite important when solving random benchmarks. Still going right in Figure 1, we encounter WATCHEDCSBJ and WALKQSAT (about 600 benchmarks solved). In a way, this result is not surprising, since WALKQSAT shares its core architecture with WATCHEDCSBJ. On the other hand, the performance of WALKQSAT is impressive if we consider that it is incomplete and, still, it can rival with its complete sibling on *mixed* instances, i.e., satisfiable and unsatisfiable ones. Back to Figure 1, QSAT stands alone with about 700 benchmarks solved. Finally, a group made of SEMPROP and QUBE-REL (resp. 802 and 810 benchmarks solved) and leaded by SSOLVE and QUBE-BJ (resp. 840 and 842 benchmarks solved) closes the listing.

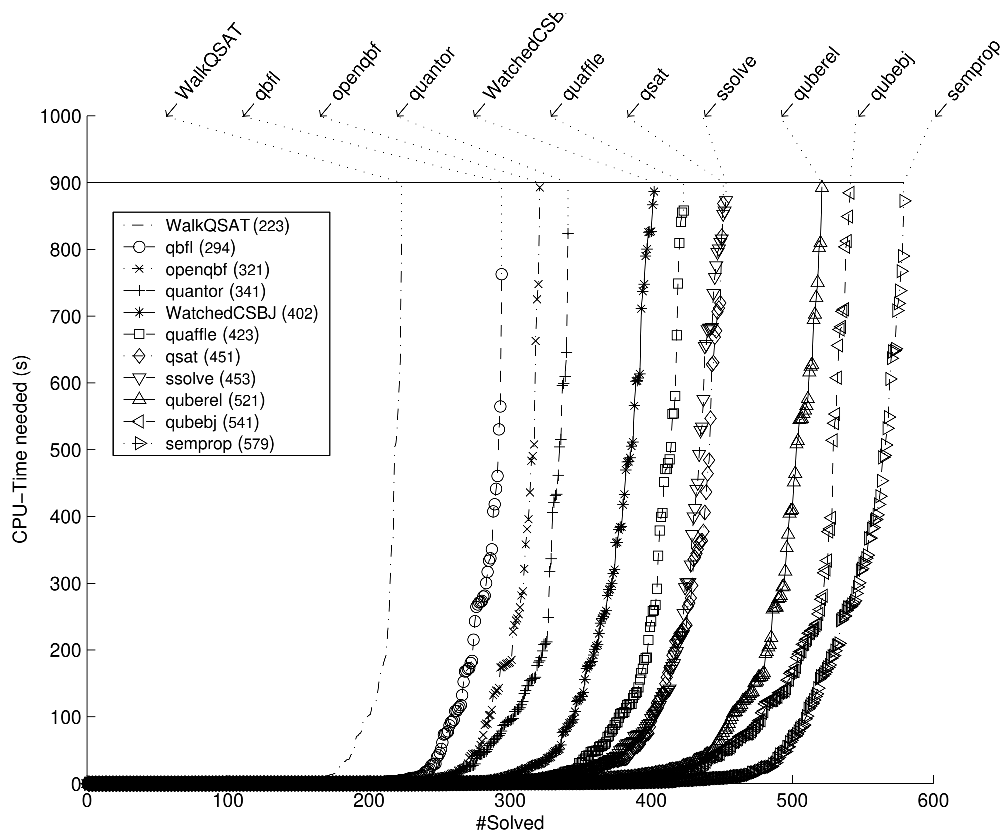

Fig. 2. Number of instances solved vs. CPU time for non-random benchmarks

Considering the results for random benchmarks, it seems that the features needed to solve effectively this kind of instances are those of QuBE-Bj and ssolve: both solvers feature traditional (counter-based) data structures, unit clause propagation, pure literal detection, trivial truth and an adaptation to QBF of Böhm's and Speckenmeyer's lexicographic heuristic as search strategy; ssolve adds to this trivial falsity, while QuBE-Bj features constraint and solution backjumping. From the evaluation data, we can conclude that pure literal detection and the particular search heuristic are playing a major role in the good performances of QuBE-Bj and ssolve. Moreover, given that random problems are generated according to model A, from the literature (see [18]), we also know that trivial truth is particularly effective on 2- and 4-QBFs, while backjumping is particularly effective on 3- and 5- QBFs. We do not know of a similar analysis for trivial falsity, but looking at ssolve performances, also this technique seems quite effective for random benchmarks. Furthermore, if we contrast the performances of semprop and QuBE-Rel with those of ssolve and QuBE-Bj, we can conclude that learning does not seem effective on random problems. This is evident if we restrict the comparison to QuBE-Rel and QuBE-Bj which differ only for the look-back engine. Finally, considering the performances of qsat, the gap of 150 benchmarks w.r.t. ssolve tells us that, besides learning, also partial unfolding, quantifier inversion and sampling do not seem to be particularly effective on the class of random benchmarks.

Non-random benchmarks Figure 2 presents the results on non-random benchmarks arranged in the same way as Figure 1. Going from left (worse) to right (best), we first encounter WALKQSAT with 223 benchmarks solved. Since its sibling WATCHEDCSBJ obtained a good standing also on non-random benchmarks, we conjecture that the problem with WALKQSAT on this kind of problems might lie in the Walksat oracle not informing the search in an effective way. Still going right in Figure 2, we see a cluster comprised of QBFL, OPENQBF and QUANTOR (resp. 294, 321 and 341 problems solved). It is interesting to notice that QUANTOR works much better on non-random, possibly because these problems have much less entropy than random ones [26]. On non-random benchmarks, a basic implementation of the universal quantifiers elimination approach, such as QUANTOR, is competitive with basic implementations of the search approach, such as QBFL and OPENQBF. Another cluster of solvers is made up by WATCHEDCSBJ, QUAFFLE, QSAT and SSOLVE (resp. 402, 423, 451, and 453 benchmarks solved) which can be considered as average performers for non-random problems. These solvers, with the exception of SSOLVE, are also SOTAC for the category of non-random problems, with QSAT being more effective than QUAFFLE and WATCHEDCSBJ (resp. 22, 9 and 2 instances uniquely conquered). Since QSAT features inversion of quantifiers, sampling and partial expansion, we can conjecture that these techniques are particularly well suited for some of the benchmarks that we used for the evaluation, while solvers of similar strength like QUAFFLE and WATCHEDCSBJ, are less effective due to their general-purpose look-back optimizations. Finally, QUBE-REL, QUBE-BJ, and SEMPROP (resp. 521, 541 and 579 instances solved) close the listing with the best performances on non-random benchmarks. In particular, SEMPROP stands out for being able to solve the highest number of benchmarks and for being SOTAC on non-random problems with 58 uniquely solved instances.

A few comments on these results are in order. First of all, the three solvers that conquered more than 500 instances implement both conflict and solution backjumping. SEMPROP and QUBE-REL add to this conflict and solution learning. While backjupimg seems to be definitely useful for non-random benchmarks, the bad performances of QUBE-REL seem to point out that learning might be ineffective on these benchmarks. This conjecture seems to be reinforced by the not-so-positive performances of QUAFFLE which features conflict and solution learning as well. Nevertheless, the evidence on QUAFFLE cannot be considered as conclusive, since this solver does not feature some of the optimizations (e.g., pure literal, trivial truth) that are found in QUBE-BJ, QUBE-REL and SEMPROP. Moreover, SEMPROP itself implements learning, while QUBE-REL learning engine does not feature fancy data structures and carefully engineered optimizations. In conclusion, non-random problems seem to favor look-ahead (sampling, inversion of quantifiers, expansion) techniques and lightweight look-back techniques (backjumping). However, the effectiveness of learning, and the trade-offs between look-ahead and look-back remain largely to be explored in this context.

3.3 Overall analysis

Strength of solvers In Figure 3 we present yet another view of the competition. In the Figure, for each solver we draw a line which represents the distance of the solver from the ideal one, i.e., one that solves all the benchmarks used for the evaluation: the longer the line, the worse the performances of the given solver. Distance is on the x-axis, solvers are listed on the y-axis; the listing is done by grouping together solvers that conquered analogous sets of instances (e.g., not surprisingly, QUBE-BJ and QUBE-REL solve a lot of instances in common). The number of benchmarks solved is showed on the line

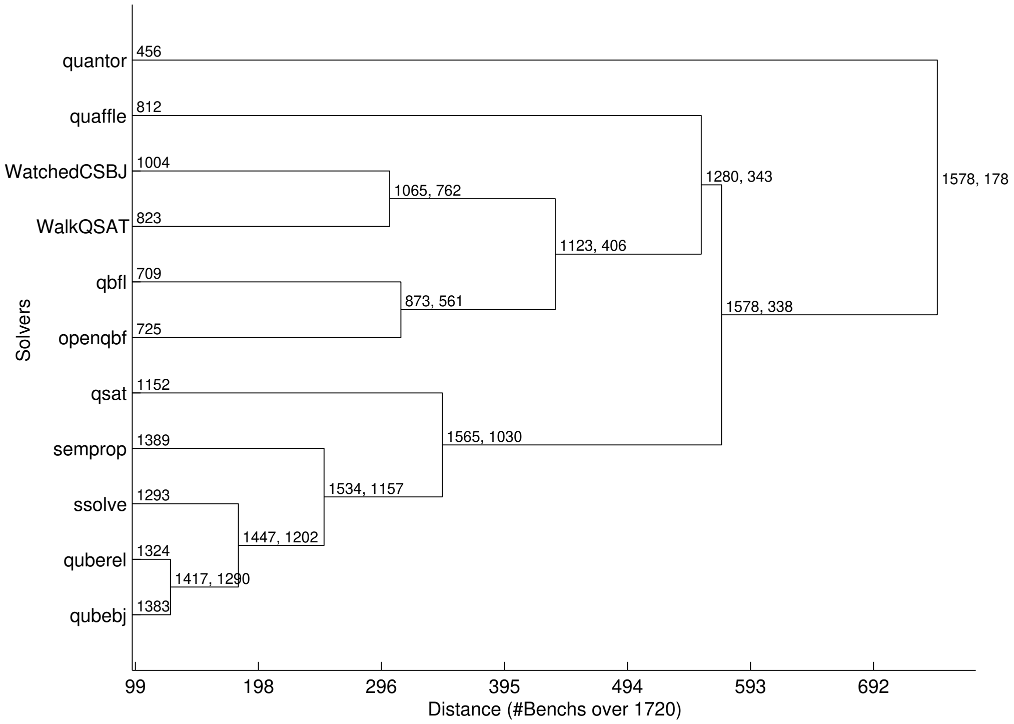

Fig. 3. Clustering of the solvers according to their ability to solve the benchmarks.

representing the solver, and the clustering of solvers is represented by two lines joining. For each cluster we indicate the union of the instances solved by the members of the cluster, and the intersection thereof (e.g., running QuBE-Bj and QuBE-Rel in parallel we could solve 1417 benchmarks, but the benchmarks solved by both of them are only 1250).

Looking at Figure 3 we can divide the solvers that participated to the evaluation into 3 groups. quantor is the only member of the first group: its peculiar architecture makes quantor rather difficult to assimilate with the other participants. Then we have a group of *medium strength* solvers comprised of openQBF, qbfl, walkQsat, watchedCSBj (SOTAC) and quaffle (SOTAC). We grouped together openQBF with qbfl, since they both represent very basic solvers (about 700 benchmarks solved), and watched-CSBj with walkQsat, since they share a common architecture (overall, walkQsat is slightly worse than watchedCSBj). The cluster is completed by quaffle, which brings the total number of benchmarks solved by this group to 1280. Finally, there is the group of *strong* solvers, composed by qsat, semprop, ssolve, QuBE-Rel and QuBE-Bj. All of these solvers are SOTAC: clustered together, they can solve 1565 problems, and around 1000 problems can be solved by every solver in this group.

Summing up, the ideal solver comprised of all the solvers running in parallel (SOTA solver in [34]) would solve 1578 benchmarks out of the 1720 used for the evaluation (a success rate of about 90%). QuBE-Bj and semprop turn out to be the strongest solvers of the evaluation (resp. 1383 and 1389 benchmarks solved), QuBE-Bj being better suited for random benchmarks and semprop better suited for non-random ones, although their performance on both categories is respectable. Conflict and solution backjumping, pure

Benchmark	#benchs	#Easy	#Medium	#Hard
Scholl/Becker	64	5	14	25
Guoqiang Pan	378	19	61	72
Ayari	72	0	15	25
Castellini	169	115	30	0
Letz	14	2	8	0
Narizzano/Random	836	16	663	7
Narizzano/Robot	120	0	36	12
Rintanen	67	21	25	0

Table 4. Hardness of benchmarks.

literal rule, trivial truth and an informed heuristic look like the key features to build a strong QBF solver. All the other features (learning, quantifier inversion, sampling, expansion) are definitely useful since the solvers implementing them are SOTAC, but a deeper analysis is needed to understand their effectiveness as general purpose techniques when used alone or in combination with other optimizations.

Hardness of benchmarks Considering the numbers of Figure 3, all the solvers were able to conquer 178 *easy* benchmarks, while 142 *hard* instances remain still unsolved. Since 1030 problems can be conquered by all the strong solvers, this leaves 862 (1030 instances minus the 178 easy ones) *medium* instances. The smallest hard instance is `blif_0.10_0.20_0_0_out_exact` in the `C432` family of benchmarks submitted by Scholl and Becker [1], having 568 variables, 3 alternations, and 1439 clauses. The biggest problem that has been solved is the `ripple-carry-15-c` instance in the `VonNeumann` family of benchmarks submitted by A. Ayari [2], having 1311754 variables, 3 alternations and 1923754 clauses. Indeed, this "monster" QBF could be solved by QUANTOR, and by the medium strength solvers OPENQBF and QBFL. Among the strong solvers, only QuBE-Rel and SEMPROP conquered the instance. Finally, Table 5 gives more details about the hardness of benchmarks, showing for each series the total number of benchmarks (#benchs) divided up into easy (#Easy), medium (#Medium), and hard (#Hard) instances. Notice that the bulk of hard instances is coming from submitted benchmarks (developers could not train their solvers on these problems) and from applications: formal verification (25 Scholl/Becker and 25 Ayari instances), planning (12 Narizzano/Robot) and modal logic encodings (72 Guoqiang Pan instances), the latter being the most challenging category in terms of absolute numbers.

4 Challenges in the QBF arena

In this Section we briefly summarize some of the open issues that occurred to us while evaluating the solvers and analyzing the results. We believe these issues will challenge developers and researchers in the QBF arena for the next few years.

Benchmarks We grouped here two open points that concern the use and the generation of benchmarks.

Challenge 1 *Solve the 142 hard QBF benchmarks remained unsolved during the evaluation.*

The first, and main, contribution of the evaluation was to isolate the instances that are too hard to be solved given the current state of the art. These instances will be made available as a single collection on QBFLIB so that developers can test and tune their solvers to conquer these problems.

Challenge 2 *Random generation model that respects the polynomial hierarchy.*

Currently the best random generators, i.e., shown experimentally to produce non-trivial formulas with controllable properties, are those based on models A and B proposed by Gent and Walsh in [16]. As noticed in previous experiments (see, e.g., [18]) and confirmed during the evaluation, formulas generated according to model A have the property of being reasonably easy in their 3,5-QBF incarnations and quite hard in their 2,4-QBF incarnations. The phenomenon is evident when using non-chronological backtracking, and it clashes with theoretical complexity results that require that 3-QBFs (resp. 5-QBFs) are harder to solve than 2-QBFs (resp. 4-QBFs). The open issue is whether a generator for random instances which is well behaved in this respect can be produced.

New techniques Challenging instances coming from demanding applications domains require the development of new techniques and the enhancement of existing ones. Here we grouped some of the interesting points that the evaluation made us think about.

Challenge 3 *Solution learning as enabling technology for new simplification techniques and heuristics.*

Solution learning in QBF has been introduced somewhat contemporaneously in [28, 29, 11]. Implementing solution learning forces us to think of the formula under test as a mixed CNF/DNF instance, the DNF part coming from the accumulation of learned solutions. This conceptual shift enables the introduction of new pruning techniques, e.g., unit clauses on universal literals (see [28]) and, possibly, new and more effective heuristics. The open point here is the exploration of the expanded possibilities that solution learning gives to developers, also considering that learning did not come out of this evaluation as an all-time winner.

Challenge 4 *Conceive and experiment new simplification and intelligent backtracking techniques.*

Although several simplification and backtracking techniques have been proposed and experimented for QBF, and that the effectiveness of solution backjumping – without a counterpart in SAT – was demonstrated during the evaluation, we still see a lot of room for improvement. Being QBF deductively more challenging than SAT, more reasoning forward (simplification) and backward (intelligent backtracking) has a good chance of paying off. The trade-off between search and reasoning must still be explored for QBF.

Challenge 5 *Conceive and experiment new effective heuristics for QBF.*

Most heuristics that have been proposed for QBF are simple adaptations of their SAT counterpart. Currently, to the extent of our knowledge, there are no comparative studies assessing the relative effectiveness of heuristics for QBF, while, judging from the evaluation results, effective heuristics might prove crucial in conquering some of the hardest problems.

Challenge 6 *Investigate the combination of technologies such as inversion of quantifiers, random sampling, partial expansion together with simplification and backtracking techniques.*

Solvers like QSAT and QUANTOR feature aggressive look-ahead techniques like quantifier expansion (resp. partial and complete), sampling and inversion of quantifiers. Others, like SEMPROP and QuBE-BJ, rely more on general purpose look-back techniques. The combination of such different philosophies, which might lead to more powerful solvers, remains largely to be investigated.

New frontiers Some interesting research trends that emerged by looking at the current state of the art.

Challenge 7 *Proving on the field that QBF-based reasoning can beat SAT-based reasoning.*

One of the reasons why SAT research flourished so much in the last decade has been the application of SAT-based encoding in several fields, e.g., planning [35] and formal verification [36] to cite only some of the most significant. Indeed, there are problems that can be solved by means of a SAT-based encoding and that admit an exponentially more compact QBF encoding. This is the case, e.g., of symbolic reachability which underlies the applications considered in [35] and [36]. Therefore the question is: is it worth using the more compact QBF encoding at the price of a more complex solver, or expanding into an equivalent (and possibly huge) SAT encoding is paying more dividends? This is a crucial question that QBF developers must answer in the next few years, one that can establish whether QBF technology is an enabling technology for real-world applications or just a theoretical exercise in automated deduction.

Challenge 8 *Build an efficient QBF solver not based on AND/OR search.*

10 out of 11 solvers participating to the evaluation are extensions to QBF of the DPLL SAT procedure. A method that works fine for SAT may not be good for QBF. QUAFFLE is a good example for that: CHAFF outperformed all other SAT solvers on many structured benchmarks when it appeared, and many current state-of-the-art solvers for structured SAT problems, if not all, inherit from CHAFF. QUAFFLE uses the very same techniques adapted to QBF, but belongs to the medium strength solvers of the evaluation. QUANTOR is a completely new approach, but we found out that basic QUANTOR is competitive with basic implementations of the DPLL procedure for QBF, which is promising in this respect.

Challenge 9 *Propose new techniques for randomized/incomplete QBF solvers.*

WALKQSAT scored quite well on the evaluation, classifying among medium strength *complete* solvers. Although it did not turn out to be SOTAC for some category, we see WALKQSAT as an interesting and promising experiment.

Challenge 10 *Build special purpose solvers, i.e., solvers tuned specifically for k-QBFs with $k \in \{2,3\}$*

Although the average number of alternations on all the benchmarks is about 9, most of the benchmarks that we used for the evaluation have a limited number of alternations (2,3) and some of the hardest problems as well. Therefore, it might be interesting to investigate methods which are specifically tuned to solve these kind of formulas, rather than arbitrary alternation depth formulas.

5 Conclusions

The final balance of this first comparative evaluation of QBF solvers is definitely positive:

- 11 solvers participated, 10 complete and 1 incomplete; this is almost twice the number of solvers that participated in the first SAT solvers competition [37], showing that research on QBF tools reached a decent level of maturity. Since most of the solvers have been proposed in recent years, the field shows a good rate of fertility too.
- 442 benchmarks were submitted coming from such diverse fields as Formal Verification [2], and Knowledge About Reasoning [5]; 1278 benchmarks were selected from more than 4000 instances available on QBFLIB [21]. These numbers witness that also the field of applications of QBF reasoning is growing steadily.
- State of the art solvers, both for random and non-random benchmarks, have been identified; also, a total of 142 challenging benchmarks that cannot be solved by none of the current QBF solvers have been identified to set the reference point for future developments in the field.
- Gathering and analyzing the data of the evaluation allowed us to compile a tentative list of future challenges in the QBF arena.

The evaluation also evidenced some critical points:

- The Q-DIMACS standard input format for QBFs in prenex clausal normal form has been around since 3 years (see [21] for a technical report about it). Nevertheless, many solvers submitted to the evaluation still used non-standard – although very close to Q-DIMACS – formats; this fact considerably complicated the work of the evaluators, by requiring the use of translation scripts;
- The numbers of the QBF evaluation are still dwarfed by the number of the SAT competition: 34 SAT solvers and 993 benchmarks from 17 different sources were submitted this year to the SAT competition.
- QBF encoding of real-world applications (e.g., Ayari's hardware verification problems, Scholl-Becker's circuit equivalence problems, etc.) contributed a lot to the pool of 142 challenging benchmarks. This shows that QBF developers must improve the performance of their solvers before these can be practical for industrial-sized benchmarks (as it is already true for SAT solvers).
- The question of how to check the answer of the QBF solvers in an effective way is still unanswered. Specifically, the questions of what is a good certificate of satisfiability/unsatisfiability for QBF, and, if this proves too huge to be practical, what is a good approximation of such certificate, remain open.

Overall, the evaluation showed the vitality of QBF as a research area. Despite some technological limitations and some maturity issues, it is our opinion that the development of effective QBF solvers and the use of QBF-based automation techniques can be regarded as promising research directions.

Thanks

The authors would like to thanks all the participants to the QBF evaluation for submitting benchmarks and solvers and the University of Genoa for providing us the computers to run the evaluation.

References

1. Cristoph Scholl and Bernd Becker. Checking equivalence for partial implementations. Technical report, Institute of Computer Science, Albert-Ludwigs University, October 2000.
2. 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.
3. Jussi Rintanen. Improvements to the evaluation of quantified boolean formulae. In *Proceedings of the Sixteenth International Joint Conferences on Artificial Intelligence (IJCAI'99)*, pages 1192–1197, Stockholm, Sweden, July 31-August 6 1999. Morgan Kaufmann.
4. C. Castellini, E. Giunchiglia, and A. Tacchella. Sat-based planning in complex domains: Concurrency, constraints and nondeterminism. *Artificial Intelligence*, 147(1):85–117, 2003.
5. Guoqiang Pan and Moshe Y. Vardi. Optimizing a BDD-based modal solver. In *Proceedings of the Nineteenth Internations Conference on Automated Deduction*, 2003. to appear.
6. Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, July 1960.
7. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.
8. Marco Cadoli, Andrea Giovanardi, and Marco Schaerf. An algorithm to evaluate quantified boolean formulae. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, pages 262–267, Madison (Wisconsin - USA), 1998.
9. Jussi Rintanen. Partial implicit unfolding in the Davis-Putnam procedure for quantified Boolean formulae. In *Logic for Programming, Artificial Intelligence and Reasoning. 8th International Conference*, number 2250 in LNAI, pages 362–376. Springer, 2001.
10. R. Feldmann, B. Monien, and S. Schamberger. A distributed algorithm to evaluate quantified boolean formula. In *Proceedings of the Seventeenth National Conference in Artificial Intelligence (AAAI'00)*, pages 285–290, 2000.
11. Lintao Zhang and Sharad Malik. Towards a symmetric treatment of satisfaction and conflicts in quantified boolean formula evaluation. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming*, pages 200–215, 2002.
12. Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. QuBE : A system for deciding Quantified Boolean Formulas Satisfiability. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR'01)*, number 2083 in LNCS, pages 364–369. Springer-Verlag, 2001.
13. Reinhold Letz. Advances in decision procedures for quantified boolean formulas. In *Proceedings of the First International Workshop on Quantified Boolean Formulae (QBF'01)*, pages 55–64, 2001.
14. Andrew G D Rowley Ian P Gent, Holger H Hoos and Kevin Smyth. Using stochastic local search to solve quantified boolean formulae. Technical Report APES-58-2003, APES Research Group, January 2003.
15. Marco Cadoli, Marco Schaerf, Andrea Giovanardi, and Massimo Giovanardi. An algorithm to evaluate quantified boolean formulae and its experimental evaluation. *Journal of Automated Reasoning*, 28(2):101–142, 2002.
16. Ian P. Gent. Beyond NP: the QSAT phase transition. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99)*, pages 648–653, 1999.
17. David G. Mitchell, Bart Selman, and Hector J. Levesque. Hard and easy distributions for SAT problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 459–465. AAAI Press, 1992.
18. Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. On the effectiveness of backjumping and trivial truth in quantified boolean formulas satisfiability. In *Proceedings of the First International Workshop on Quantified Boolean Formulas (QBF'01)*, pages 40–54, 2001.
19. *Sixth International Conference on Theory and Applications of Satisfiability Testing*, S. Margherita Ligure - Portofino (Italy), May 2003.

20. Laurent Simon and Philippe Chatalic. SATEx: a web-based framework for SAT experimentation. In Henry Kautz and Bart Selman, editors, *Electronic Notes in Discrete Mathematics*, volume 9. Elsevier Science Publishers, June 2001. http://www.lri.fr/ simon/satex/satex.php3.
21. M. Narizzano. QBFLIB - The Quantified Boolean Formulas Satisfiability Library. http://www.qbflib.org.
22. Gilles Audemard, Daniel Le Berre, Olivier Roussel, Inês Lynce, and Jo ao Marques Silva. OpenSAT: An Open Source SAT Software Project. In *Sixth International Conference on Theory and Applications of Satisfiability Testing* [19].
23. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 530–535, June 2001.
24. M. Bohm and E. Speckenmeyer. A fast parallel sat–solver — efficient workload balancing. *Annals of Mathematics and Artificial Intelligence*, 17:381– 400, 1996.
25. Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified boolean satisfiability solver. In *Proceedings of International Conference on Computer Aided Design (ICCAD'02)*, San Jose, CA, USA, Nov. 2002.
26. Armin Biere, 2003. Personal communications.
27. Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Backjumping for quantified boolean logic satisfiability. In *Proceedings of the Seventeenth International Joint Conferences on Artificial Intelligence (IJCAI'01)*, Seattle, Washington, USA, August 4th-10th 2001. to appear.
28. Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Learning for quantified boolean logic satisfiability. In *The Eighteenth National Conference on Artificial Intelligence (AAAI'02)*, Edmonton, Alberta, Canada, July 28–August 1 2002.
29. Reinhold Letz. Lemma and model caching in decision procedures for quantified boolean formulas. In *Proceedings of Tableaux 2002*, LNAI 2381, pages 160–175, Copenhagen, Denmark, 2002. Springer.
30. Andrew G. D. Rowley, Ian Gent, Enrico Giunchiglia, Massimo Narizzano, and Armando Tachella. Watched data structures for QBF solvers. In *Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 348–355, 2003. Extended Abstract.
31. Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, pages 337–343, Seattle, 1994.
32. D. Pretolani. *Satisfiability and Hypergraphs*. PhD thesis, Università di Pisa, 1993.
33. Peter Balsiger, Alain Heuerding, and Stefan Schwendimann. A benchmark method for the propositional modal logics k, kt, s4. *Journal of Automated Reasoning*, 24(3):297–317, 2000.
34. Geoff Sutcliff and Christian Suttner. Evaluating general purpose automated theorem proving systems. *Artificial Intelligence*, 131:39–54, 2001.
35. Henry A. Kautz and Bart Selman. Pushing the envelope : Planning, propositional logic, and stochastic search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'96)*, pages 1194–1201, 1996.
36. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *International Conference on Tools and Algorithms for Construction and Analysis of Systems*, volume 1579 of *LNCS*, pages 193–207. Springer-Verlag, 1999.
37. M. Buro and H. Kleine Büning. Report on a SAT competition. *Bulletin of the European Association for Theoretical Computer Science*, 49:143–151, 1993.