The essentials of the SAT 2003 competition

Daniel Le Berre[1] and Laurent Simon[2]

[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

Abstract. The SAT 2003 Competition ran in February – May 2003, in conjunction with SAT'03 (the Sixth Fifth International Symposium on the Theory and Applications of Satisfiability Testing). One year after the SAT 2002 competition, it was not clear that significant progress could be made in the area in such a little time. The competition was a success – 34 solvers and 993 benchmarks, needing 522 CPU days – with a number of brand new solvers. Several 2003 competitors were even able to solve within 15mn benchmarks remained unsolved within 6 hours by 2002 competitors. We report here the essential results of the competition, interpret and statistically analyse them, and at last provide some suggestions for the future competitions.

1 Introduction

Organizing a competition is a large effort, made by the whole community, both for submitters and organizers. It is thus essential to ensure that this effort is keeping worthwhile for the community. Clearly, such an event is keeping the community excited and allows a good visibility of the SAT field outside its own frontiers. For the second edition of the competition, the challenges were to bring up new solvers, to measure progress made since last year and, more importantly, to identify promising techniques and new challenges for SAT solvers.

The aim of this paper is to analyze the good (and bad) points of the competition, to fairly report the performances of solvers and to emphasize what we think was (or was not) a success. Exhaustive results are available via the web, in a dynamic HTML and a raw format, which allows anyone to make its own analysis. The paper is organized as follows: in a first part, after recalling the main rules of the contest, we present the solvers, the benchmarks and the results of the contest identifying the winners. Then, we analyse the first phase results and finally we discuss SAT'03 competition issues and propose some improvements for the next competition. We'll try to demonstrate that the competition is now mature enough to evolve in such way that it can fulfill its three fundations principles: motivate the field, identify new benchmarks and techniques and experimentally study solvers behavior.

1.1 Brief recall of competition rules

The competitions runs in two phases. The first one is a way to identify the best solvers and the hardest benchmarks, in order to use more time for the both of them in the second phase. The first phase is thus important to provide exhaustive empirical results for analysis, while the second one provides more limited results, mainly dedicated to award both solvers and benchmarks.

We use the notions of *categories* and *series* for benchmarks. They are partitionned into 3 categories (*Industrial*, *HandMade* and *Random*), then in series. If the idea of categories is clear, the idea of a series is to group together benchmarks that are similar (*e.g.* same generator, ...) to identify solvers that can solve a wide variety of different benchmarks. A series is considered as solved if at least one benchmark in the series is solved. The solvers are ranked after the first phase according to the number of series and then the total number of solved benchmarks. From those anonymized rankings, the competition judges, decide which solvers should enter the second phase for each category. Note that only one variant of the same solver is elligible for the second phase. After the second phase, the winner is the solver that solved the greatest number of benchmarks. The competition awards 6 solvers (3 categories of benchmarks in which we award the best complete solver (answers SAT+UNSAT) and the best solver that answered SAT (Complete + Incomplete solvers) and the smallest unsolved SAT (resp. UNSAT) benchmark. If a solver answers UNSAT on a proved (checked by a certificate) SAT instance, then that solver is declared "buggy".

The competition ran this year on two clusters of Linux boxes. One, from the "Laboratoire de Recherche en Informatique" (LRI, Orsay, France), was composed of 15 Athlon 1800+ with 1Gb memory. The second one, from "Dipartimento di Informatica Sistemica e Telematica" (DIST Genoa, Italy), was composed of 8 PIV 2.4GHz with 1Gb memory.

2 The competitors

2.1 Solvers

Solver submission closed on February 14th but late submissions were accepted *hors-concours*[1] until the first phase really started.

We had 28 **complete non randomized** solvers: `berkmin` (5.61 and 6.2, awarded last year) from E. Goldberg and Y. Novikov [10]; `bmsat` (hors concours) from X. Xu; `compsat` (0.5) from A. Biere. Because a bug was discovered, we report the fixed version, `compsat-fixed`, as a hors-concours solver; `farseer` (1.00) from M. Girard, `forklift` from E. Goldberg and Y. Novikov ; `funex` (0.2) from A. Biere; `jerusat` (1.1 *a, b, c*) from N. Alexander [2]; `jquest2` from I. Lynce and J. Marques-Silva [18]; `kcnfs` from G. Dequen and O. Dubois [8, 7]; `limmat` (1.3), awarded last year, from A. Biere; `lsat` (1.1) from R. Ostrowski,

[1] Those solvers entered the first phase, which allowed them to compete with current state-of-the-art SAT solvers, without being awardable

B. Mazure and L. Sais [22]; `march sp/tt/xq` from M. Dufour, M. Heule, J. van Zwieten and H. van Maaren. Because a bug was found in `marchxq`, we report here the fixed version, named `marchxq-fixed`, as a hors-concours solver; `oepir` (0.2) from J. Alfredsson; `oksolver` from O. Kullmann [13] as an *invited* hors-concours solver (`oksolver` was awarded twice last year); `opensat` (0.44) from G. Audemard, D. Le Berre and O. Roussel [3]; `satnik` from N. Sörensson [9, 6]; `sato` (3.4) from H. Zhang [28]; `satzilla v1/v2/v2-fixed` (0.9) (v2s are hors-concours) from K. Leyton-Brown, E. Nudelman, G. Andrew, C. Gomes, J. McFadden, B. Selman and Y. Shoham [16, 15]. Authors discovered during the contest that `satzilla2` had bugs in its learning phase that might lead to wrong choices of solvers. The corrected version is refered as `satzilla2-fixed`; `satzoo v0/v1` (0.98) from N. een [9]; `tts` (1.0), hors-concours, from I. Spence; and `zchaff` from Y. Yu and L. Zhang [21, 29], awarded last year.

We had 2 **complete randomized** solvers: `siege`, hors concours, from L. Ryan and `xqinting` (0.1) from X. Yu Li, M. Stallmann and F. Brglez.

And, at last, we had 4 **incomplete** (one-sided SAT) solvers: `amvo` from S. vogt and A. Machen; `qingting` (1.0) from X. Yu Li, M. Stallmann and F. Brglez [17]; `saturn` (2) from S. Prestwich [23]; and `unitwalk` (0.981) from E. Hirsch and A. Kojevnikov [11].

2.2 The benchmarks

We had 17 submitters, but a lot of benchmarks were submitted by the same people (BMC from IBM, generators for random/structured formulae). We also reused all SAT'02 challenges (except some from SAT'02 random challenges, because there was too many of them). We tried to equilibrate the cpu effort on categories: Industrial with 323 benchmarks in 45 series, Handmade with 353 benchmarks in 34 series, Random with 317 benchmarks in 30 series. Like last year, we shuffled all the benchmarks to avoid easy syntactical recognition of benchmarks by solvers. Regarding random and handmade benchmarks, we only considered 3 benchmarks per point. That means that once we fixed all the parameters for the generator, we generated 3 (and only 3) benchs with 3 different seeds for that particular point.

The new benchmarks are, for the **Industrial** category (we have a total number of 323 benchmarks with, on average, 42703 variables, 163947 clauses and 452283 literals): `hard_eq_check` (16 benchs in 1 serie), from E. Goldberg; `addm` (6 benchs, 1 serie) from J. kukula (equivalence check of two randomly structured adders with multiple addends); `li-exam` and `test` (12 benchs, 2 series) from R. Li (unit covering problem of logic minimization of logic circuit benchmark test4); `ferry`, `gripper` and `hanoi` (24 benchs, 3 series) from F. Maris (based on TSP (tunable satplan), a generator for planing problems in PDDL format); `l2s` (11 benchs, 1 serie) from V. Schuppan (Bounded model checking of Bwolen Yangs collection of benchmarks for SMV in conjunction with a method to convert liveness to safety); `zarpas` (223 benchs in 25 series) from E. Zarpas (a lot of (large) BMC formulae).

For the **Handmade** category, we have a total number of 353 benchmarks having, on average 3233 variables, 37355 clauses and 154485 literals: `sgi` (120 benchs, 8 series) from C. Anton (subgraph isomorphisms); `graphs` (56 benchs, 7 series) from R. Bevan (based upon various graphs as first described by Tseitin in 1968); `qgh` (19 benchs, 2 series) from C. Gomes (quasigroups with holes); `chesscolor` (5 benchs, 1 serie) from F. Lardeux (encode an original chessboard coloring problem); `multLin` and `mm` (17 benchmarks, 1 serie) from K. Markstrom (various instances of fast matrix multiplication); `genurq` (10 benchs, 1 serie) from R. Ostrowski (modified Urquhart problems, all satisfiable); `clusgen` (9 benchs, 3 series) from A. Slater (randomly generated clustered 3SAT problems; `hwb` (21 benchs, 1 serie) from T. Stanion (equivalence checking problem of two different circuits computing the hidden weighted bit function).

We mainly restricted the **Random** category to benchmarks that *looked like* uniform random benchmarks. We have 317 benchmarks with on average 442 variables, 4164 clauses and 25401 literals: `balanced` and `hidden` (111 benchs, 8 series) from C. Moore (1 or 2 hidden assignments and balanced literals); `hgen*` (57 benchs, 4 series) from E. Hirsch (generators based on `hgen2`, last year winner); `gencnf` (63 benchs, 7 series) from G. Dequen (uniform generator at treshold for $k \in [4, 10]$); `uniform` (45 benchs, 5 series) from L. Simon (uniform generator for k=3, one serie for each ratio $r \in \{3.0, 4.0, 4.25, 4.5, 5.5\}$).

3 First phase results

The first phase began on March 19th and ended on April 14th. During the whole process of the competition, solver submitters were able to check virtually in real time the progress of their own solvers. All the traces were available, and suspicious launchs (due to hardware/network error, ...) were ran again. The two solvers `compsat` and `marchxq` were found incorrect during the first phase due to minor bugs. Let us recall that we present here the result of their fixed version, `compsat-fixed` and `marchxq-fixed`. We will not discuss here the results of the SAT subcategory in both the industrial and handmade categories, since there are not enough benchmarks to draw any conclusions.

3.1 First phase on Industrials

The figure 1 illustrates the results of the complete solvers on all the benchmarks of the industrial category. This representation, used in the previous competition report [26], allows to check the choice of the cpu timeout and to have clues about solvers behaviors on all the benchmarks in a given category. Last year winners in the category are `zchaff` and `limmat`. Those solvers end a group of zchaff-like solvers leaded by `forklift`. If we take those solvers as the references for state-of-the-art SAT solver in 2002 then it seems clear that some progress was made in the industrial category.

But the picture is not so clear. Let us take a look at `berkmin62 curve`: One can note that not that many solvers are stronger than `berkmin62`, which enterred

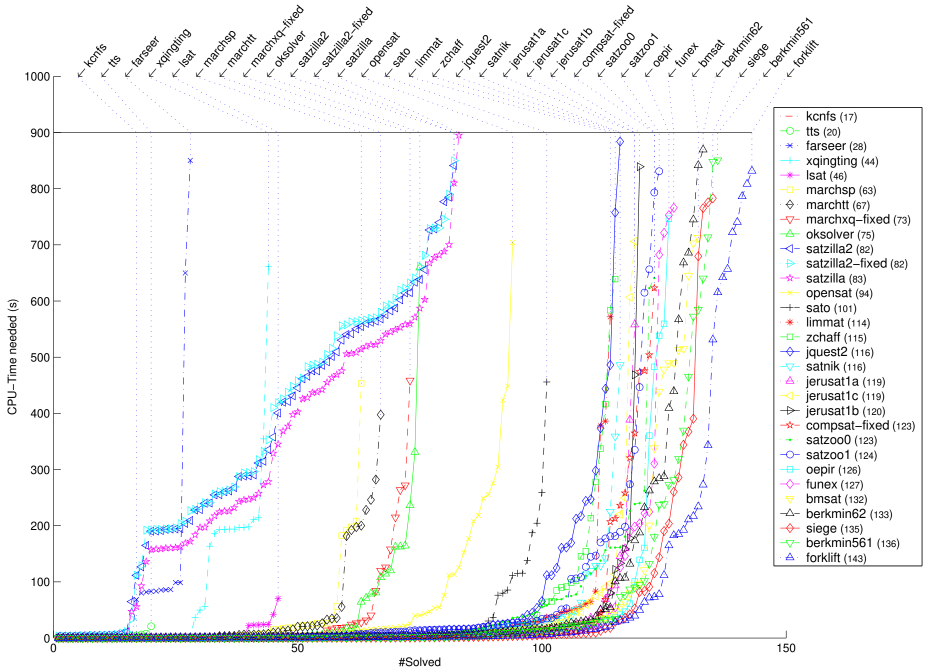

Fig. 1. # of solved instances vs. CPU time for complete solvers on all industrial benchs

the SAT'02 competition but had a bug that prevented it to solve hundreds of benchmarks [26]. The current version of `berkmin62` is of course fixed, so it is reasonable to consider it as a progress witness for the year 2002. Now, the progress made is not that important: only three solvers performed better that `berkmin62` and two of them are from the same authors, one solver being a variant of one of the engine of `berkmin62`, while the other one is a new solver that can be seen as an extension of `berkmin62` with binary clause reasoning.

At last, one can notice that there is many zchaff-like solvers this year (13 solvers, 17 variants, against 4 solvers last year). This can be explained by the huge interest in the practical resolution of SAT in the EDA community and by the fact that zchaff was a breakthrough in the community in the last two years. So, the good point is that a lot of new solvers were proposed and showed relatively good performances for their first public evaluation. But the pernicious effect of the competition is that, in the run for good performances, many solvers are very similar (zchaff-like).

3.2 First phase on Handmade

The figure 2 illustrates the results of the complete solvers on handmade benchmarks. One solver, `lsat`, clearly shows a particular behavior. This solver doesn't implement only resolution, but can handle (even long) chains of equivalency

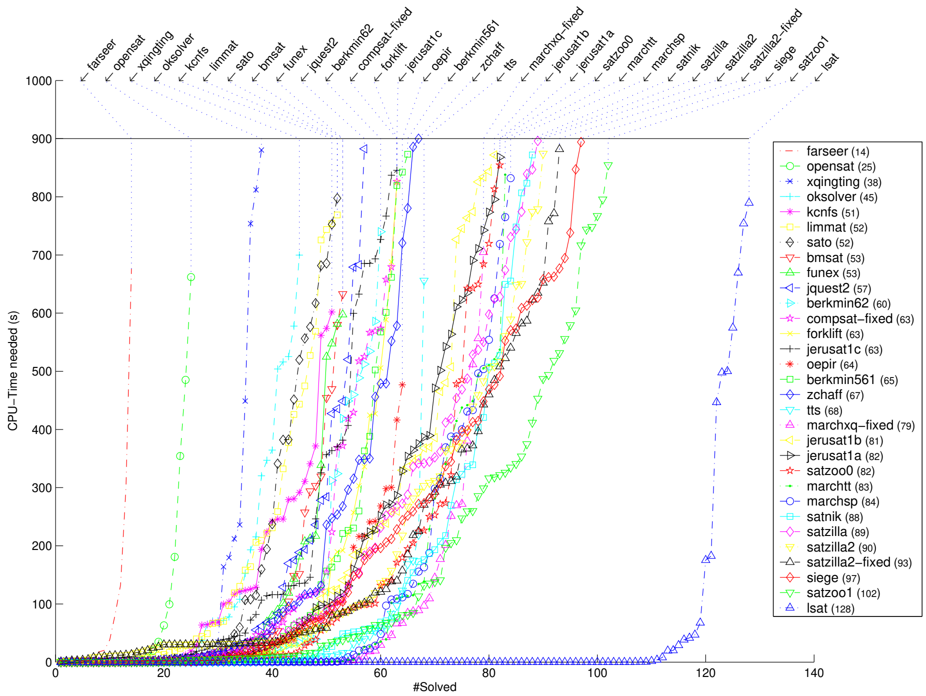

Fig. 2. # of instances solved vs. CPU time for complete solvers on all handmade benchs

clauses and can more generally retrieve from a CNF some boolean function on which it applies some simplifications [22]. That point gives it the ability to quickly solve tricky formulas, commonly used in this category. The figure also shows relatively efficient solvers between `marchxq-fixed` and `satzoo1` on this category.

3.3 First phase on Random

Results for the two random categories are given on figures 3 and 4. On the first figure, one can easily identify all complete solvers good for random instances (between `oksolver` and `satzilla2-fixed`). The curves have a relatively slow growth (certainly due to the fact that random instances scales smoothly), especially for `kcnfs`, which seems to scale well. This is not the same picture for `satzilla`, which present the same kind of curve than `lsat` on HandMade: a lot of benchmarks are solved quickly, and then the curve have an important growth. On SAT instances, `unitwalk` and `satzilla2` present the same kind of curves.

The case of satzilla The solver takes an unusual, "portfolio" approach to the SAT problem [16, ?]. It uses machine learning models to choose the best solver among a set of existing SAT algorithms. For `satzilla` 0.9, these include: `2clseq`,

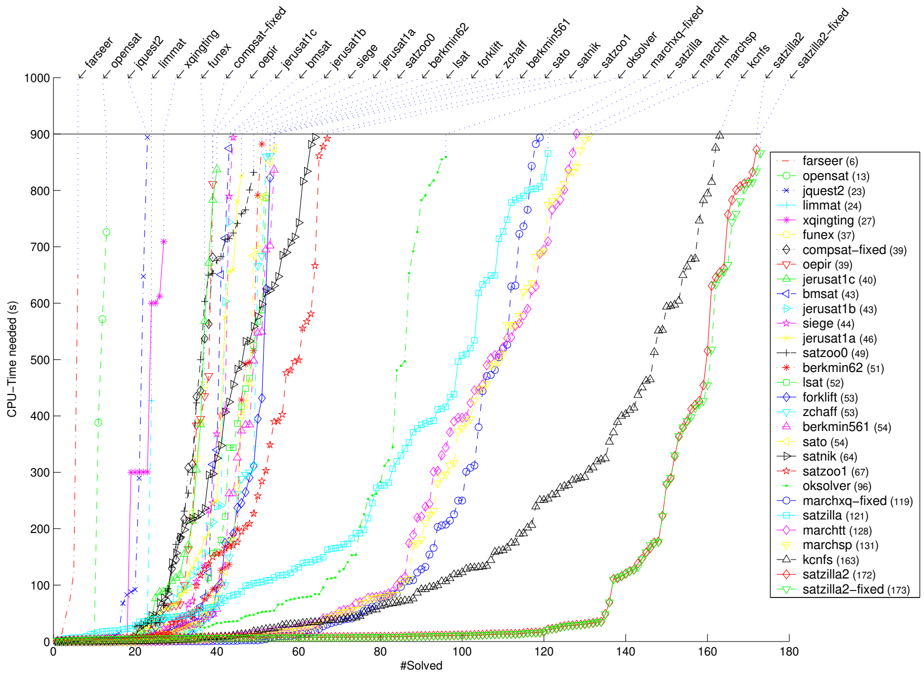

Fig. 3. # of instances solved vs. CPU time for complete solvers on all random benchs

`limmat`, `jerusat`, `oksolver`, `relsat`, `sato`, `satzrand`, and `zchaff` (Hors Concours solver `satzilla2` uses two additional complete solvers, `eqsatz` and `heerhugo`). It also executes the `autoSat` algorithm for a short time before starting any other computation, and is thus able to filter out easy satisfiable problems (this point can explain the kind of curves we observed). We conducted a few analysis on `satzilla2-fixed` and report how many times it launched a given solver and how many times it solved the benchmarks (number in parenthesis). On industrial instances, it launched `jerusat` 13 (3) times, `zchaff` 64 (49) times, `eqsatz` 1 (0) time, `2clseq` 27 (4) times, `satzrand` 4 (0) times and `limmat` 2 (0) times; For HandMade, the choices were `jerusat` 93 (42), `relsat` 1 (0), `eqsatz` 26 (6), `2clseq` 5 (0), `satzrand` 39 (9), `limmat` 62 (1), `oksolver` 26 (1), `heerhugo` 7 (0) and `sato` 10 (3); at last, on random instances, the choices were `jerusat` 1 (1), `zchaff` 12 (2), `eqsatz` 18 (18), `2clseq` 13 (1), `satzrand` 124 (37) and `oksolver` 35 (1). The choices were different for each category and one may notice that `zchaff` was choosen 12 times on random instances (and even gave 2 results).

Note that `eqsatz` was launched 18 times on random problems and succeeded each time. This is because the `hardnm` benchmarks from C. Moore contains equivalency chains that can be solved in linear time by specific solvers such as `eqsatz` of `lsat`. For such a reason, the behavior of `satzilla2` on those “random” benchmarks must be taken with care.

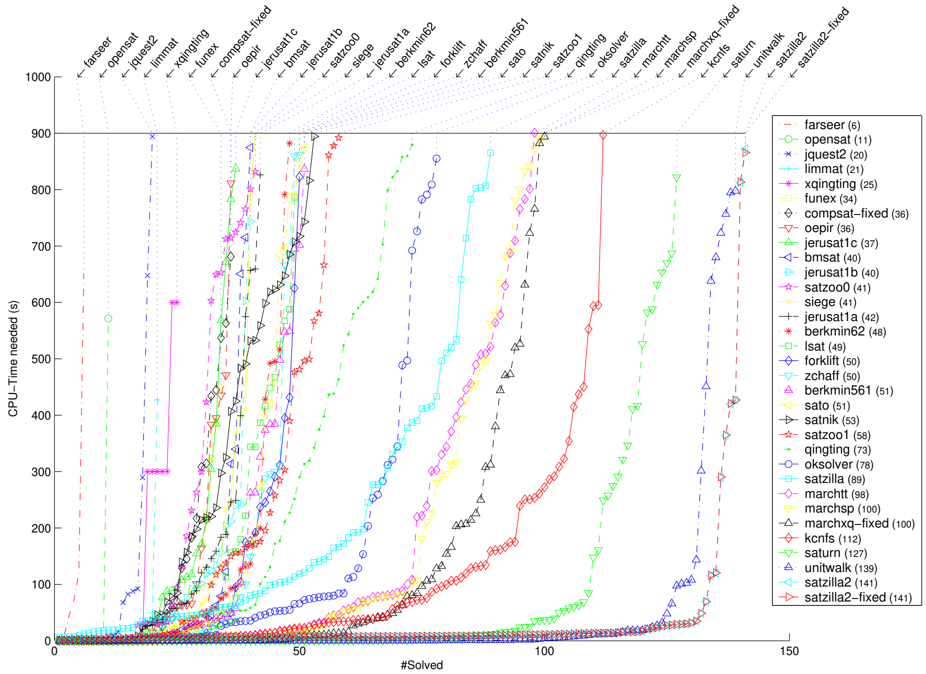

Fig. 4. # of instances solved vs. CPU time for all solvers on SAT random benchs

3.4 Solvers entering the second phase

This year, the three judges decided, from an anonymized version of the rankings presented table 1 which solvers should enter the second phase of the competition. Table 1 also emphasizes the solvers that were alone to solve at least one benchmark (so called State-Of-The-Art Contributor, SOTAC, in [27]) during the first phase. In addition to the one reported in the tables, `marchxq-fixed` (Industrial) and `compsat-fixed` (Industrial, HandMade) are also SOTAC.

3.5 The hardest benchmarks

The pool of benchmark remained unsolved during the first phase is available on the web. We encourage developpers to focus on this set of hard benchmarks. We have 174 benchmarks (from 16 series) in the Industrial category (mean (min–max) sizes: #clauses=222811 (7814–3113540), #variables=61322 (2269–985042), #literals=627908 (21206–7976612)), 165 benchmarks (from 18 series) for the HandMade one (#clauses=54422 (832–630000), #variables=4518 (75–270000), #literals=263992 (2818–1350000)) and 91 (from 17 series) for the Random category (sizes: #clauses=7838 (391–36000), #variables=286 (38–700), #literals=61515 (888–360000)). Among the unsolved benchmarks, we often find large industrial formulas and, in the random category, k-cnf with large k.

Solvers	#series	#benchs (comments)	Solvers	#series	#benchs (comments)
Industrial					
Complete solvers on all benchs			All solvers on SAT benchs		
forklift	34	143 (SOTAC)	forklift	9	33
berkmin561	32	136	berkmin62	8	32
satzoo1	31	124	zchaff	8	31
oepir	30	126	oepir	8	31
funex	29	127 (SOTAC)	jerusat1b	8	31
satnik	29	116	funex	8	31
jquest2	29	116	satnik	8	30
zchaff	29	115 (SAT'02 winner)	limmat	8	30 (SAT'02 winner)
jerusat1b	28	120	satzoo1	7	30
limmat	28	114	jquest2	7	29
HandMade					
Complete solvers on all benchs			All solvers on SAT benchs		
jerusat1b	22	81	satzoo1	11	37 (SOTAC)
satzoo1	21	102	jerusat1b	11	35
satzilla	21	89	satzilla	11	25
satnik	20	88	forklift	10	30 (SOTAC)
marchsp	19	84	oepir	9	31 (SOTAC)
lsat	15	128 (SOTAC)	satnik	9	30
			berkmin561	9	30 (v62 was SAT'02 winner)
			saturn	7	29 (SOTAC)
Random					
Complete solvers on all benchs			All solvers on SAT benchs		
kcnfs	25	163 (SOTAC)	kcnfs	18	112
satzilla	19	121	unitwalk	15	139 (SOTAC)
marchsp	18	131	saturn	14	127
oksolver	16	96	satzilla	14	89
satnik	13	64	marchsp	13	100

Table 1. Ranking of solvers entering the second phase.

4 The Second phase: the winners

4.1 Best solvers

The table 2 summarizes the whole second phase of the contest. Here, the cpu timeout was set to 2 hours (instead of 15mn). The 4 winners are in bold face in each category. Briefly, **forklift** won both (SAT+UNSAT and SAT) Industrial categories, **satzoo1** won both HandMade categories. **kcnfs** won the Random SAT+UNSAT category and **unitwalk** the Random SAT category.

Results are not surprising for `forklift`, according to the first phase. It is also encouraging to see that `kcnfs` won a prize, because that may be a good indicator that our partitioning may be not so bad (`kcnfs` was devoted to solve such instances). One can also notice that this is the first time an incomplete solver (`unitwalk`) winning the competition. Last year, `oksolver` won both the Random categories prizes. `oksolver` was even not qualified for the second phase on Random SAT instances (it was on SAT+UNSAT).

At last, we have to point out that winners are often determined by very small difference. This is not really surprising since the shapes of curves are almost vertical on figures 1–4 for these solvers, but may be disappointing since after one month of contest, the winner was determined on its performances on a very few benchmarks.

Solver	#benchs	Solver	#benchs	Solver	#benchs
Industrial Complete		Industrial SAT		HandMade Complete	
forklift	**12**	**forklift**	**4**	**satzoo1**	**9**
berkmin561	11	jerusat1b	3	lsat	7
satzoo1	5	satnik	3	satzilla	6
jerusat1b	5	funex	1	satnik	3
satnik	4	zchaff	1	jerusat1b	1
zchaff	4	oepir	1	marchsp	0
funex	3	berkmin62	1		
oepir	1	satzoo1	1		
jquest2	0	limmat	0		
limmat	0	jquest2	0		
Random Complete		Random SAT		HandMade SAT	
kcnfs	**12**	**unitwalk**	**12**	**satzoo1**	**5**
satzilla	6	kcnfs	4	satzilla	3
oksolver	4	saturn	2	forklift	3
marchsp	3	satzilla	2	berkmin561	1
satnik	0	marchsp	0	saturn	0
				oepir	0
				satnik	0
				jerusat1b	0

Table 2. Ranking of solvers after the second phase

4.2 Hardest benchmarks

In each categories, we selected the smallest benchmarks to use during the second phase, according to their number of literals, in order to award the smallest unsolved benchmarks in both SAT and UNSAT categories.

The winner for the smallest unsat benchmark is `random/hirsch/hgen8/hgen8--n260-01-S1597732451`, a benchmark submitted by E. Hirsch. It was renamed `sat03-885` during the contest. This unsatisfiable benchmark contains 391 clauses, 260 variables and only 888 literals. In comparison to last year results, the smallest UNSAT instance moved from the HandMade to the Random category, which can be explained by two reasons. First, `lsat` has droped away, during the first stage, all small benchmarks based on equivalency clauses and second, the category of this benchmarks may not be the good one.

On the SAT category, we didn't award any benchmark. The winner was `random/simon/sat02-random/hgen2-v400-s161064952` (renamed `sat03-1681` this year), a benchmark submitted by E. Hirsch last year. It contains 1400 clauses, 400 variables and 4200 literals. That benchmark was solved in the second phase last year, but not this year. Such a phenomenon may be due to bad luck this year for randomized instances, or simply to the re-shuffling of the instance. We'll discuss this kind of problem in the section 5.2.

5 Discussion

5.1 Other rankings

To check the strongness of the competition final results or to fin other ways to analyse the results, one may wants to try other methods to rank solvers and to compare the rankings. For instance, in the contest, we tried to forget about

Solver	Total Time (s)	#solved benchs	Solver	Total Time (s)	#solved benchs
Complete on Industrials			Complete on HandMade		
forklift	171240	143	lsat	207371	128
siege	175168	135	satzoo1	241664	102
berkmin561	176088	136	satnik	249446	88
bmsat	178445	132	siege	250043	97
berkmin62	179121	133	marchtt	250711	83
Complete on Random			All On SAT Random		
satzilla2	147208	172	satzilla2	35402	141
kcnfs	165655	163	unitwalk	38572	139
unitwalk	166372	139	saturn	52136	127
saturn	179936	127	kcnfs	67493	112
marchsp	192688	131	marchsp	83030	100

Table 3. Top-5 ranking of solvers *à la satex*

the cputime, by counting only the number of series and benchmarks solved. Of course the cpu time had a direct impact on the competition due to the timeout, but it is important to try to characterize efficient solvers, in term of cpu time.

Ranking *à la* **SatEx** In SatEx [25], rankings are done by counting the total cpu time needed for one solver to solve the whole category of benchmarks, using the timeout as the default penalty. Note that the timeout was 15 mn, which is not a big penalty, so a fast solver can easily be ranked above a solver that solved more benchmarks. Table 3 summarizes the top-5 solvers in each category that we studied in this paper. `forklift` is still a clear winner (`siege` was hors-concours) on Industrial benchmarks. On HandMade benchmarks, such a ranking would have awarded `lsat` instead of `satzoo1`. `lsat` was able to quickly solve a large amount of benchmarks without search, but when the DLL engine of `lsat` was needed (on hard instances like in the second phase), `lsat` was not able to beat `satzoo1`. On random benchmarks, `satzilla2` (which is hors-concours) would be ranked first, which is in part due to the 18 `hardnm` benchmarks outlined before.

Relative efficiency of solvers One of the hard things to handle for ranking solvers is that only partial informations are available, because of the timeout. One possible way is to compare only pairs of solvers (X,Y) on the subset of benchmarks they both solves (let use write $s(X)$ as the set of benchmarks solved by X), then we can compare X and Y on their respective performances on the set $s(X) \cap s(Y)$). When doing this, we have a strong way of comparing the *relative efficency* (RE) of X and Y : $re(X,Y) = s(X) \cap s(Y)/s(Y)$ gives the percentage of instances of Y that X solves too. Let us write now $cpu(X,b)$ the cpu time needed for X to solve all the benchmarks in b, without any timeout. Because there was a timeout in the competition, only $cpu(X,s')$, with $s' \subseteq s(X)$ are defined here for the solver X. Now, we can compare the relative efficiency of X with Y by computing $crr(X,Y) = cpu(X, s(X) \cap s(Y))/cpu(Y, s(X) \cap s(Y))$. This last measure is called here the *cputime reduction ratio* (CRR), and means that, on their common subset of solved benchmarks, X needs only $crr(X,Y)$ percent of the time needed by Y. Now, to summarize all the possible values, we average these two measures over all the possible values for Y, while keeping X fixed, and we thus defined $re(X)$ and $crr(X)$, two new measures to compare solvers performances.

In the Industrial (complete) category, we have the following values for re : `forklift` (98.9%), `berkmin561` (98.7%), `siege` (98.6%), `berkmin62` (98.4%), `oepir` (97.6%), `funex` (97.4%), `bmsat` (97.2%), `compsat-fixed` (96.8%), `satzoo0` (96.4%) and `satzoo1` (96.3%). The crr values are `forklift` (23%), `siege` (28%), `berkmin561` (46%). `funex` and `oepir` needs only 55% of the time of the other solvers. The crr values of solvers then jumps to more than 70% for the other solvers. It is interesting to remark that both `forklift` and `siege` are very efficient solvers ($crr < 25\%$ and an impressive value for re). `satzoo0` is also interesting because it has a high re and a crr at 164%. This is the same kind of remark for `jquest2`, which has a re of 94.5% and a crr at only 388% (certainly due to the java penalty). In a sense, we think that such a measure may emphasize a robust solver (if $s(X)$ is large enough): `zchaff` has a re of 91.1% and a crr of 87%, and `limmat` a re of 93.9% and a crr of 138%. Thus, `jquest2` and `limmat` are slower than `zchaff`, but can on average solve more benchmarks.

Interestingly, in the Random (SAT) category, we have the following re: satzilla2 (92.8%), satzilla2-fixed (92.8%), marchxq-fixed (87.0%), marchsp (82.2%), marchtt (80.1%) and kcnfs (78.9%). The best crr are different here: unitwalk, have only a re of 71.4% but a very remarkable crr of 6%! saturn has a re of 71.1% and a crr of 28%. That means that unitwalk and saturn solve problems different from other solvers, and solve the common problems very quickly. In the order, after these two incomplete solvers, the crr are sazilla2-fixed and satzilla2 (29%) and kcnfs (81%). The crr then falls to more than 1OO%. In some sense, unitwalk present the same kind of results than lsat: a low re value that indicates here an original method. If $crr < 1$ then this method is fast, and, at last, the method is *interesting* if $s(x)$ is large enough. This is clearly the case for lsat and unitwalk.

5.2 The *lisa syndrom*

It was brought to the fore in [17] that solvers may have drastically different behaviors on a range of problems generated from one original benchmark by randomly shuffling[2] it. This year, the same benchmark (a SAT'02 challenge called *lisa*) was placed in two different categories by error. Before we detected the problem, it was shuffled twice and solvers ran on both instances. Here are some examples of solvers that showed different behaviors on those two benchmarks ("–" means *unsolved*): `jerusat1a` (–, 40.71) ; `jerusat1b` (236.85, –) ; `oepir` (–, 12.62) ; `siege` (67.43, –), `satzoo1` (–, 287.12) and `satzilla` (196.8, 727.52). If such a result can question the validity of the competition, it may be smoothed by the large number of benchmarks used. This is not the case for the second phase. Considering the small number of benchmarks solved, beeing lucky can make the difference.

[2] The shuffler used for the competition generates members of the *PC-class* of [17].

5.3 Progress or not?

One of the most important questions when assessing the results of the competition was whether or not substantial progress was made. Our first thought was "yes", since several benchmarks unsolved during 6 hours last year were solved within 15 mn this year. But, let's take a closer look at each category. Progress on the Industrial category was already discussed in section 3.1. In the handmade category, `lsat` shows very good performances on many structured benchmarks. However, `lsat` participated last year and became hors concours because of a bug. This year version is mainly last year version without the bug. So it is hard to see here any real progress since last year. In the random category, the picture is a bit different. Some new solvers were submitted this year. Last year winner, `oksolver`, was outperformed in the satisfiable category. The most interesting thing concerning that category is the presence of the hors concours `satzilla2`, a portfolio solver, that compared favorably against the awarded complete solver and did slightly better that the incomplete solver awarded for satisfiable benchmarks.

5.4 Benchmark categories: back to the roots

Some improvements were made concerning the way solvers were evaluated this year: only the best variant of a given solver was eligible for the second phase, solver submitters were able to follow almost in real time their solver during the first phase of the competition, the judges chose the solvers that should enter the second phase, which was more flexible and fair than using an arbitrary rule (top 5 solvers..), etc. But nothing really changed regarding the way we handled benchmarks. We need to focus on this issue for the next year competition.

The idea of creating three categories of benchmarks, namely industrial, handmade and random, was to cover the different interests of researchers in the practical resolution of SAT. The industrial category was aimed to address the practical resolution of SAT problems for SAT-based reasoning engines. Ideally, that category should reflect the strengths or weaknesses of the solvers as a SAT component, in a "real/industrial" setting. Bounded model checking[5, 4] or more generally all the applications from the Electronic Design Automation community[19], planning[12], cryptography[20] were good candidates for such category. Up to now, we received mainly benchmarks from the EDA community. Those benchmarks really reflected the kind of problems to be solved by an embedded SAT engine. Unfortunately, since we shuffled the benchmarks, we did not really evaluate the solvers in "real conditions". The order of the clauses, the identifiers of the variables may provide some structural information about the initial problem and some solvers built to solve those kind of problems were designed to handle that information. By shuffling the benchmarks, we hide that information. Thus a solver can have a very different behaviour during the competition and "in situ", which is in contradiction with one of the aims of the contest. We have three solutions. First, we keep all things like this and we let submitters explicitly write heuristics in their solver that take shuffling into account. If this allows to emphasize the existence of implicits heuristics based on the syntactical order of the formula (which is a good point), this kind of solution is weird in some sense: solvers will spend time trying to rearrange the formula! We can also avoid shuffling for this category, but then solvers may be too much tuned for benchmarks, casting doubts on the results on challenging, publicly available, benchmarks. The last choice may be to launch the solvers on the original benchmark and X shuffled benchmarks of the same benchmarks (this solution may also adresses the lisa syndrom).

The random category was designed to evaluate the progress made on a class of theoretically well known and hard (NP-complete) problems: random k-sat. There are not that many ways to generate random k-SAT problems, so most of the benchmarks used for that category where based on some uniform k-SAT generators. However, we received some generators able to build problems looking syntactically like random uniform problems, but are forced SAT or UNSAT (like all the generators from Edward Hirsch). The question is: were those generators in the right category? Even worst is the case of the hardnm benchmarks related previously. Theoretical and practical work on random k-SAT is based on uniform k-sat. Furthermore, some problems such as quasigroup with holes, or clusters of random k-SAT formulas were in the handmade category. So the random category would better be renammed “uniform random” category and all the non uniform random generators should move to another category. Wether that category should be a new one or should be the handmade category is not yet decided.

The handmade (crafted) category was aimed to point out new techniques not based on resolution. Finding an efficient technique to tackle SAT problems not based on resolution is one of the ten propositional challenges proposed in [24]. That category was also a reminder that if some solvers can solve huge benchmarks in the industrial category, some small benchmarks are still out of reach of those solvers. The solvers awarded so far in that category (zChaff, Berkmin and satzoo) are using resolution. So the category is awarding very efficient solvers (typically with the greatest number of decisions per second) rather than clever ones. Note that this year, an hybrid solver recovering potential structural knowledge and using it to simplify the initial problem solved the greatest number of problems in that category, but was not awarded. Awarding the most efficient solver is of interest, pointing out “clever” solvers would be a plus for the competition.

At last, we observed in both industrial and handmade category very few satisfiable benchmarks. One way to solve this problem is to filter satisfiable benchmarks from a pool of benchmarks submitted to the competition. In that case the benchmarks are needed long before the solvers, which means to change the way we benchmarks are submitted. Another solution is to ask benchmarks submitters a balanced number of SAT and UNSAT benchmarks.

6 Summary of proposals for SAT'04 competitions

Despite the huge amount of data collected during the contest, it is impossible to really understand why a solver is better than another. In order to empirically study solvers (e.g. find hypothesis on their behaviour, ...), we will probably ask submitters next year to output statistics about their runs. XML output format could be great for automated analysis of results (each submitter may then provide its own ontology) but we are thinking of something even simpler. Such information may lead to identify the solver that explore the smallest search space, the fastest solver on unit-propagation, ... and then to begin to really understand experimentally what's going on.

Now, one of the problems this year was to choose the right category for all benchmarks. A lot of different solutions exist. First, one could simply use the huge amount of data in the first phase to automatically (*e.g.* statistically) clusterizes the benchmarks into different categories. This is for instance done by the authors of satzilla that computes 60 polynomial time features per instance to decide which solver to use on which benchmark. However, we may have a large number of different categories and we think that is very usefull to keep the process in the hands of human. An automatic partitionning would imply that a given bench may change of category from one year to another one. Moreover, the original notion of categories is somewhat well understood and corresponds to some expert knowledge about benchmarks. Another solution would be to ask for the help of judges to classify benchmarks, according to some description given by the author.

How to "officially" point out new (promising) methods? The quest for fast-solvers is the pernicious effects of the contest and we have seen that this year, not so much new methods have been proposed. Because one of the aim of the contest was to promote new clever methods, it could be nice to have some kind of special prizes, given by the judges to any solver, based on the analysis of the first phase results. Of course, this solver may not be the fastest and the more efficient one, but should be the most "promising" one.

The second phase is also a point which may be questionned. We have seen that winners may simply be lucky on a very few instances. In some sense, it is frustrating to spend years of cpu time and to determine the winner on its performances on less than 5 instances (where luck can take a large part). Moreover, all the important data are gathered during the first phase. So, could we reconsider the second phase? We are thinking of two alternative solutions: determine the winner according to its performances on the first phase or using some kind of tournament in the second phase, where each step would ensure a clear winner between 2 solvers.

Briefly, a lot of other points may be discussed before next year competition:

- To begin the competition earlier, just discard solvers not complying with the input/output requirements.
- Do not run 3 times randomized solvers since deterministic solvers can be lucky too (cf the lisa syndrom).

– Collect benchmarks in a different way: we need more control over benchmark submission in order to ensure a meaningful competition.
– Allow benchmarks submitters to follow "live" the solvers on their benchmarks.

The most important issue for next the next competition will be the notion of classes of benchmarks instead of just one instance, to obtain strong result on solvers performances. A reasonable solution is to run the solvers on the original benchmarks (without shuffling them) and on a set of X shuffled version of that benchmarks, X depending on the number of benchmarks, solvers and computers available. Such approach provides a view of the solvers "in situ" and assesses their robustness.

7 Conclusion

The competition was exciting and worthwhile. We presented in this paper a number of different views and analysis based on the data gathered during the first phase. Many new solvers were submitted this year, a majority of them were zchaff-like solvers. There was no outstanding improvements compared to last year, but a reasonable set of strong solvers for "industrial" benchmarks is now available. Forklift, which inherits from berkmin62, leads those solvers. In the handmade category, lsat demonstrated an interesting behavior (a former version participated to SAT'02 but was incorrect). Satzoo showed better performances in terms of diversity of benchmarks and scalability. In the random category, the warded solvers are not a surprise: the solver kcnfs in the complete category, the incomplete solver unitwalk in the satisfiable category.

The most interesting result of the competition is the behavior of the portfolio algorithm satzilla, which compared favourably with the awarded solvers in the random category. Wether this result can be replicated in the other categories is an interesting challenge.

Next year, a new competition will be organized in conjunction with SAT2004, in Vancouver. We showed in this paper that the competition is now mature enough to fulfill three main goals: (1) motivate the field, (2) promote new solvers and hard benchmarks and (3) learn from the observation of solvers behaviors.

Acknowledgements

We especially want to thanks our three judges, John Franco, Hans van Maaren and Toby Walsh for their involvement in every phase of the competition. Authors would also like to thank the "Laboratoire de Recherche en Informatique" (LRI, Orsay, France) and the "Dipartimento di Informatica Sistemica e Telematica" (DIST Genoa, Italy) for providing us with clusters of machines. At last, we thank all the authors of solvers and benchmarks for their participation and their effort to make their solver I/O compliant with our format. Without them there would be no competition at all!

References

1. *Sixth International Conference on Theory and Applications of Satisfiability Testing*, S. Margherita Ligure - Portofino (Italy), May 2003.
2. Nadel Alexander. Backtrack search algorithms for propositional satisfiability: Review and innovations. Master's thesis, Hebrew University of Jerusalem, November 2002. please email alikn@hotmail.com to receive an electronic copy of this work.
3. 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* [1].
4. A. Biere, A. Cimatti, E.M. Clarke, O. Strichman, and Y. Zhu. *Highly Dependable Software* , volume 58 of *Advances In Computers*, chapter Bounded model checking. Academic Press, 2003.
5. Armin Biere, Alessandro Cimatti, Edmund M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of bdds. In *Proceedings of Design Automation Conference (DAC'99)*, 1999.
6. Koen Claessen and Niklas Srensson. Finite model generation for first order logic using propositional satisfiability. Submitted for publication.
7. Gilles Dequen and Olivier Dubois. Renormalization as a function of clause lengths for solving random k-sat formulae. In *Proceedings of Fifth International Symposium on Theory and Applications of Satisfiability Testing*, pages 130–132, 2002.
8. Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3-sat formulae. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, Seattle, Washington, USA, August 4th-10th 2001.
9. Niklas Een and Niklas Srensson. Temporal induction by incremental sat solving. Submitted for publication.
10. E. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT-solver. In *Design, Automation, and Test in Europe (DATE '02)*, pages 142–149, March 2002.
11. E. A. Hirsch and A. Kojevnikov. UnitWalk: A new SAT solver that uses local search guided by unit clause elimination. PDMI preprint 9/2001, Steklov Institute of Mathematics at St.Petersburg, 2001. A journal version is submitted to this issue.
12. 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.
13. O. Kullmann. Heuristics for SAT algorithms: Searching for some foundations, September 1998. 23 pages, updated September 1999 w.r.t. running times, url = http://cs-svr1.swan.ac.uk/ csoliver/heur2letter.ps.gz.
14. Daniel Le Berre, Laurent Simon, and Armando Tachella. Challenges in the QBF arena: the SAT'03 evaluation of QBF solvers. In *Proceedings of SAT2003*, 2003. submitted.
15. K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. A portfolio approach to algorithm selection. In *Proceedings of IJCAI'03*, 2003. A full version of this paper is available from http://robotics.stanford.edu/ kevinlb/boosting.pdf, and is under review.
16. K. Leyton-Brown, E. Nudelman, and Y.Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Proceedings of CP'02*, 2002.
17. X. Y. Li, M.F. Stallmann, and F. Brglez. QingTing: A Fast SAT Solver Using Local Search and Efficient Unit Propagation. In *Sixth International*

Conference on Theory and Applications of Satisfiability Testing [1]. See also `http://www.cbl.ncsu.edu/publications/`, and `http://www.cbl.ncsu.edu/-OpenExperiments/SAT/` .
18. Inês Lynce and Jo ao Marques Silva. On implementing more efficient data structures. In *Sixth International Conference on Theory and Applications of Satisfiability Testing* [1].
19. Joo P. Marques-Silva and Karem A. Sakallah. Boolean Satisfiability in Electronic Design Automation. In *Proceedings of the IEEE/ACM Design Automation Conference (DAC)*, pages 675–680, June 2000.
20. Fabio Massacci. Using walk-sat and rel-sat for cryptographic key search. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 290–295, Stockholm, Sweden, July 31-August 6 1999. Morgan Kaufmann.
21. 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.
22. R. Ostrowski, E. Grégoire, B. Mazure, and L. Sais. Recovering and exploiting structural knowledge from cnf formulas. In *Proc. of the Eighth International Conference on Principles and Practice of Constraint Programming (CP'2002)*, LNCS, pages 185–199, Ithaca (N.Y.), September 2002. Springer.
23. S. D. Prestwich. Randomised backtracking for linear pseudo-boolean constraint problems. In *Proceedings of Fourth International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems*, 2002.
24. Bart Selman, Henry A. Kautz, and David A. McAllester. Ten challenges in propositional reasoning and search. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 50–54, 1997.
25. 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.
26. Laurent Simon, Daniel Le Berre, and Edward E. Hirsch. The sat2002 competition report. *Annals of Mathematics and Artificial Intelligence*, 2003. Special issue for SAT2002, to appear.
27. Geoff Sutcliff and Christian Suttner. Evaluating general purpose automated theorem proving systems. *Artificial Intelligence*, 131:39–54, 2001.
28. Hantao Zhang. SATO: an efficient propositional prover. In *Proceedings of the International Conference on Automated Deduction (CADE'97), volume 1249 of LNAI*, pages 272–275, 1997.
29. L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient conflict driven learning in a Boolean satisfiability solver. In *International Conference on Computer-Aided Design (ICCAD'01)*, pages 279–285, November 2001.