Fifty-Five Solvers in Vancouver: The SAT 2004 Competition

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

[1] CRIL-CNRS FRE 2499, 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. For the third consecutive year, a SAT competition was organized as a joint event with the SAT conference. With 55 solvers from 25 author groups, the competition was a clear success. One of the noticeable facts from the 2004 competition is the superiority of incomplete solvers on satisfiable random k-SAT benchmarks. It can also be pointed out that the complete solvers awarded this year, namely Zchaff, `jerusat1.3`, `satzoo-1.02`, `kncfs`and march-eq, participated in the SAT 2003 competition (or at least former versions of those solvers). This paper is not reporting exhaustive competition results, already available in details online, but rather focuses on some remarkable results derived from the competition dataset.

The SAT 2004 competition is ending a 3-year take-off period that attracted new SAT researchers and provided many new benchmarks and solvers to the community. The good participation rate of this year (despite the addition of the anti-blackbox rule) establishes the competition as an awaited yearly event. Some new directions for a new way of thinking about the competition are discussed at the end of the paper.

1 Introduction

Building efficient SAT solvers is one of the many sides of SAT research. Whether these solvers are built as a proof of concept for a theoretical result, or are the result of a careful software engineering process for industrial purposes, they are useful for the whole community because they provide a snapshot of current algorithmic performances. Efficient SAT solvers help us estimate which SAT instances are solvable on current computers, and which methods works on which kind of problems.

As we've done in the previous competitions, we partitioned the set of benchmarks in three categories. The recent use of SAT solvers as embedded engines in successful model checkers or planning systems[1] created a huge interest in building efficient SAT solvers especially dedicated to solving SAT benchmarks with thousands (sometimes millions) of variables. These very large benchmarks are coming from an automated translation of problems from bounded model checking [2], formal verification [23], planning [11], etc. They may be found in the *industrial category* of the competition: they provide an "optimistic" bound of the kind of problems solvable by current state of the art SAT solvers. However, if many solvers have been reported to solve most of these large industrial benchmarks today, it is still possible to build a three-hundred-variable benchmark that they won't be able to solve. Most of these benchmarks arise from the theoretical result of NP-completeness of the satisfiability problem. They may be found in the *crafted category* of the contest. Such category often provides a "pessimistic" bound on the size of brute force solvable problems and may also emphasize new inference rules for strengthening SAT solvers. At last, the uniform random k-CNF – formulas containing exactly k different literals per clause – is still a widely used class of benchmarks, both useful at the theoretical and the practical level. These benchmarks represent a very particular but still hard challenge (especially the unsatisfiable random benchmarks) for the SAT community. We classified them in the *random category*.

As previously, the competition was based on a two-stage ranking. The deadline for submitting both solvers and benchmarks was February, 23rd. Solvers ran first on randomly generated random k-SAT, then on industrial benchmarks. The first stage finished early April by running the solvers on the remaining crafted instances. The authors of the SAT solvers were able to follow almost in real time the progress of their solver, ensuring the correctness of the collected information. The timeout for the first stage was only 10 minutes, because of the large number of solvers competing this year. After this first stage, the solvers were ranked for each category according to the number of solved series, then according to the total number of solved benchmarks[2]. The aim of this ranking is to focus on SAT solvers able to solve a wide range of SAT benchmarks. From an anonymized version of those rankings the judges, João Marques-Silva, Hans Kleine-Büning and Fahiem Bacchus, decided for each category which solvers were eligible to enter the second stage. Note that the solvers for which a detailed description was not available were not eligible to enter the second stage.

These "second stage solvers" were then launched on the benchmarks that remained unsolved with a longer timeout (40 minutes). Technically, 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 Athlon 1800+ with 1 GB memory. It was used last year for the SAT 2003 competition.

[1] SATPLAN04, powered by the SAT solver Siege, got the first place in the 2004 planning competition in the optimal track `http://ls5-www.cs.uni-dortmund.de/~edelkamp/ipc-4/results.html`.

[2] A series of benchmarks is a collection of similar benchmarks (e.g. for the random category, benchmarks with the same number of variables and the same number of clauses). A series is solved if at least one of its benchmarks is solved.

The second one, from "the beta lab" (UBC, Vancouver, Canada), was composed of Intel Xeon 2.4 GHz with 1 GB memory.

2 The competitors

2.1 Solvers

Due to lack of space, we'll not describe here each solver. We invite the reader to take a look at the 2-page descriptions of the solvers available on the competition results web page[3]. 25 submitters (see the online solver descriptions for the co-authors) provided 55 solvers:

- **Complete (SAT and UNSAT) solvers**:
  `brchaff` (R. Bruni), `circush0`, `circush1` (H. Jin) [9], `cls` (W. Ruml) [6], `compsat` (A. Biere), `cquest` (I. Lynce), `eqube1`, `eqube2` (M. Narizzano), `forklift` , `frankensat-high` `frankensat-low` (M. Dufourt) , `funex` (A. Biere), `isat1`, `isat2`, `isat3` (N.S. Narayanaswamy), `jerusat1.3` (A. Nadel), `lsatv1.1` (R. Ostrowski) [15], `march-001`, `march-007`, `march-eq-010`, `march-eq-100` (M. Heule), `minilearning.jar` (D. Le Berre), `modoc` (A. Van Gelder) [21], `nanosat` (A. Biere), `oepira`, `oepirb`, `oepirc` (J. Alfredsson), `ofsat` (O. Fourdrinoy), `quantor` (A. Biere) [1], `sato4.2`, `sato4.3` (H. Zhang) [27], `satzoo-1.02` (N. Een), `tts-2-0` (I. Spence), `werewolf` (J. Roy), `wllsatv1` (R. Ostrowski), `zchaff`, `zchaff-rand` (Z. Fu) [14],
- **Incomplete (SAT) solvers**:
  `adaptnovelty`, `novely35`, `novely50` (H. Hoos)[8], `gasat`(F. Lardeux) [10], `qingting` (X.-Y. Li) [13], `rsaps`, `saps`, `sapsnr` (D. Tompkins), `saprover`, `unitwalk` (A. Kojevnikov) [7], `walksatauto`, `walksatrnp` [8] and `walksatsck` [17] (H. Kautz).
- **Portfolio** (contain both complete and incomplete solvers):
  `satzilla`, `satzilla-nr`, `satzilla-r` (E. Nudelman)

The solvers `forklift` (E. Goldberg and Y. Novikov) and `kncfs` (Gilles Dequen and Olivier Dubois [5]) were not submitted by their authors but entered the contest as last year winners.

2.2 The benchmarks

Uniform random k-SAT 300 benchmarks were generated in 30 series: 15 series with 3-SAT benchmarks, 15 series with k-SAT formulas ($k > 3$). For the 3-SAT series, 3 series of most probably SAT (ratio number of clauses over number of variables equals 4) and UNSAT (ratio 4.5) benchmarks for 500, 700 and 900 variables were generated. The remaining 9 series were generated at the threshold (ratio 4.25) for 400 to 800 by 50 variables. For the 15 k-SAT benchmarks, $k = 4, 5, 6, 7, 8$, the instances were generated at the threshold for

[3] `http://www.lri.fr/~simon/contest/results/`

3 different number of variables (different for each k). Let us emphasize that the number of variables used were adapted from last year solvers performances: we needed instances not too easy but not too hard to discriminate the solvers, considering only last year solvers performances.

Industrial During last year competition, some competitors spotted that their solvers were behaving much better on the original industrial benchmarks than on the shuffled version we were using in the competition. The order of the clauses and the numbering of the variables were supposed to have a meaning in those benchmarks and some solvers might be able to take that information into account. As a consequence, the performance of their solver during the competition did not match their observation in an industrial setting. We took this remark into account and generated 3 series from one in this category: one with the original benchmarks, and two with different shuffling of the original benchmarks.

The industrial category benchmark set was composed of 157 original benchmarks in 18 series. Two of them were dedicated to benchmarks that remained unsolved since the first competition. Another 7 formed the benchmarks not solved last year. The 10 new series were proposed by:

Hoonsang Jin 8 benchmarks from BMC using the Vis system [16].
Marijn Heule 1 benchmark from Philips
Allen Van Gelder 12 benchmarks encoding into SAT coloring problems [20].
Miroslav Velev Formal Verification problems in two series, pipe-sat-1.1 [22] (10 benchmarks) and vliw-unsat-2.0 (8 benchmarks).
Emmanuel Zarpas 5 series from IBM BMC benchmarks [25]: 01-rule, 11-rule-2, 13-rule, 22-rule and 30-rule.

Crafted Called "Hand Made" in the previous competition, this category contains 29 series for a total of 228 benchmarks. Four series contained instances unsolved since the first competition (`hgen, ezfact, urquhart, others`), 9 series contained benchmarks that remained unsolved last year (`bqwh, chesscolor, factoring, anton SAT/UNSAT, hirsch, moore, markstrom UNSAT, station hwb`). The new benchmarks were submitted by:

Andrew Slater two series of randomly generated clustered benchmarks.
Hoonsang Jin one Ramsey benchmark.
Ke Xu 2 series from forced satisfiable CSP instances of Model RB [24].
Marijn Heule 2 series of equivalency chains.
Calin Anton 4 series of Random l-clustered k-SAT.
Harold Connamacher 5 series of benchmarks encoding the generic uniquely extendible constraint satisfaction problem [3].

Note that this year the crafted category contains the "look like random series" `hgen+, hirsch, moore` that belonged to the random category in the previous competition.

3 First stage results

3.1 First stage on Random

The first obvious comment about the results of the first stage in the random category is that the incomplete solvers outperformed the complete ones this year. A nice way to see it is to take a look at Figure 1. Incomplete solvers appear on the right (from `saps` to `saprover`): they solve quickly a large number of benchmarks. Complete solvers are on the left. The complete local search solver `cls` lies in between. The `satzilla`'s solvers that use a local search solver as a preprocessing step are located with the local search solvers.

Note that the best incomplete solver in number of series solved was able to solve all the series containing SAT benchmarks (3 series of 3-CNF were generated to be UNSAT). While all the incomplete solvers were able to solve at least 100 benchmarks, the best complete solver `cls` was only able to solve 65 of them. Note that last year winner `kncfs` was only able to solve 60 benchmarks. So it looks like we have a good new complete SAT solver for the random category this year? Not really. `cls` is a complete local search solver: compared to its local search siblings, it performs poorly on satisfiable benchmarks. Compared to its complete siblings, it perform poorly on UNSAT problems (it did not solve one!).

Another obvious fact is that none of the conflict-driven clause-learning algorithms extending `zchaff` (including the original) succeeded to enter the second stage. None of them was able to solve an UNSAT instance. The best of them, with respect to the number of benchmarks solved, `satzoo-1.02`, was only able to solve 13 instances in the series `k3-r4.25-v400` (9), `k3-r4-v500` (3), `k4-r9.88-v155` (1). `satzoo-1.02` solved the first series in 2500 s while the incomplete solvers solved it in less than a second and specialized complete solver such as `kncfs` and `march` needed respectively 38s and 18s.

The three very strong solvers `saps`, `walksatrnp` and `adaptnovelty` are similar to each other. Basically, `walksatrnp` and `adaptnovelty` are very similar in the sense that they are two implementations of the Rnovelty+[8] version of Walksat. The difference lies in the value of the noise (p): `walksatrnp` uses a fixed value of $p = 0.5$ while `adaptnovelty` adapts the value of p during the search. `saps` is using a different approach (scaling and probabilistic smoothing) but is implemented in the same framework as `adaptnovelty`: UBCSAT[4]. The series solved by `walksatrnp` but not solved by `adaptnovelty` and `saps` is `k7-r88.7-v110`, the collection of 10 7-CNF with 110 variables at ratio 88.7. `gasat` also failed to solve that series. Note that `gasat` solved many series but significantly fewer benchmarks than the three best solvers. This may be explained by either a greater algorithmic complexity or a less efficient implementation of the solver. This tends to be confirmed in the second stage (Table 2) since `gasat` solved 17 new benchmarks while most of the other incomplete solvers solved only a few (2 to 7), by the shape of `gasat` in Figure 1 and its *crr* value of 1.82 (Detailed explanation of this value is given later, when table 5 is introduced).

On the UNSAT instances, only a limited number of series and benchmarks were solved. As a result, the ranking of the overall category (SAT+UNSAT) is quite close to that of the SAT category.

At the satisfiability threshold, the generated instances have the same probability to be SAT or UNSAT, thus one could expect half of the instances to be SAT and the other ones UNSAT. During the first stage, exactly 150 benchmarks were proved SAT by the incomplete solvers, which is exactly half of the benchmarks. In order to check this result, after the competition, we launched `kncfs` and `adaptnovelty` with a 15-hour timeout, on the still-unsolved random benchmarks, and they were unable to find new SAT ones (`kncfs` found 18 new UNSAT benchmarks). Even if such a result may look nice, let us notice that the number of SAT/UNSAT benchmarks was not uniformly dispatched across the series.

3.2 First stage on Industrial

The industrial category is a very competitive category, mainly because of its economic aspect. Some industry-related solvers competed in the previous competitions but authors of last year winner, `forklift`, declined the invitation to submit a detailed description of their solver in the conference post proceedings, due to intellectual property reasons. If this can be easily understood from their point of view, from the organizers point of view, and with the agreement of the judges, it was decided not to use the SAT competition to publicize a solver without providing anything back to the SAT community.

We introduced for that reason the “Anti-Blackbox rule” that prevents SAT solvers for which the details are not known (no source code available, no publicly available technical report about it) to participate in the competition. This is a clear path on which we want the competition to stay. For those reasons, the popular SAT solver `siege`[4] did not enter the contest this year, but `forklift` entered the contest as last year winner. The `oepir` solvers entered the competition, an overall description of the solvers being provided, but a more detailed description was borderline with a non-disclosure agreement. For that reason, the solvers became ineligible for an award. The following results do include both `forklift` and `oepirs` ones, because we think that awarding a solver and reporting its results are different things. *It is important for the community to see where those solvers stand compared to the other ones.*

Without any surprise, Conflict-Driven Clause-Learning (CDCL) solvers outperform the other ones in that category. All the solvers entering the second stage either include or extend a CDCL solver. The more robust solver is the last year winner `forklift`, which solved 30 series for a total of 81 benchmarks. The best solver with respect to the total number of benchmarks solved is a variant of Oepir, `oepirc`, which solved 24 series for a total of 95 benchmarks. The series solved by `forklift`and not by `oepirc` are Schuppan `l2s` and Miroslav Velev `VLIW-UNSAT-2`, for a total of 8 benchmarks. Note that `oepirc` was able to solve

[4] http://www.cs.sfu.ca/~loryan/personal/

them neither under their original form nor after shuffling. Both series contain huge benchmarks (15 000 to 1M variables for `l2s`, 25 000 to 500 000 variables for `VLIW`).

`oepirc` outperformed `forklift` on IBM benchmarks, especially on the original series. `zchaff-rand` is also a strong solver with 27 series and 67 benchmarks solved. It was not able to solve the `l2s` benchmarks. `brchaff`, `circush1`, `compsat` and `jerusat1.3` were able to solve more than 50 benchmarks. On the one hand, `oepirc`, `jerusat1.3` and `compsat` solved significantly more SAT benchmarks than the other solvers, but on the other hand, `oepirc` and `forklift` outperformed the other solvers in the UNSAT category, followed by `zchaff-rand`. Interestingly, `jerusat1.3` was the worst performer in the UNSAT category (among the second stage solvers).

Twice as many UNSAT than SAT series were solved, same ratio for the number of benchmarks solved, which emphasizes strong UNSAT solvers in the overall ranking.

3.3 First stage on Crafted

The results in this "everything not uniform random or industrial" category are quite close. The incomplete solver `sapsnr` was the most robust solver with 11 series solved in the satisfiable category, but all the second stage solvers were quite close: there is only a difference of 8 benchmarks solved between the first and the last solver entering the second stage. In the UNSAT category, the most robust solver with 7 series solved is `satzoo-1.02`, while the strongest solvers with respect to the number of benchmarks solved are `march-007` and `march-eq-010`. On the overall ranking, `march-eq-010` and `satzilla-nr` are the most robust solvers while `march-007` and `march-eq-100` remain the best solvers with respect to the number of benchmarks solved. It is worth noting that in the crafted category, the three top solvers in the overall ranking are using completely different technologies.

4 The Second Phase: The Winners

Table 2 summarizes the result of the second stage in the three categories. The solvers `oepirc` and `forklift` do not appear since they were not awardable. During the second stage, the solvers were ranked according to the number of benchmarks solved *among the benchmarks remained unsolved during the first stage*.

Winners in the industrial category are consistent with the results of the first stage (once `forklift` and `oepirc` have been discarded): `zchaff-rand` is awarded in the industrial UNSAT category while `jerusat1.3` is awarded in the industrial SAT category. Since the number of UNSAT benchmarks solved is greater than the number of SAT benchmarks solved, the overall ranking emphasizes strong UNSAT solvers. As a results, `zchaff-rand` is also the winner in the industrial overall category.

ALL			SAT			UNSAT		
INDUSTRIAL								
Solver	#series	#benchs	Solver	#series	#benchs	Solver	#series	#benchs
Forklift	30	81	CQuest	10	25	Forklift	24	51
zchaff_rand	27	67	CirCUsH1	10	23	zchaff_rand	18	40
brchaff	25	58	OepirC	9	40	brchaff	17	29
CirCUsH1	25	53	Jerusat1.3	9	39	OepirC	16	55
OepirC	24	95	compsat	9	36	Satzoo_1.02	16	18
compsat	23	57	Forklift	9	30	CirCUsH1	15	30
CQuest	22	45	brchaff	9	29	quantor	15	21
minilearning.jar	22	44	zchaff_rand	9	27	Jerusat1.3	15	17
Jerusat1.3	21	56	minilearning.jar	8	25	compsat	14	21
quantor	21	37	Satzoo_1.02	7	22	minilearning.jar	14	19
Satzoo_1.02	19	40	quantor	6	16	CQuest	12	20
CRAFTED								
Solver	#series	#benchs	Solver	#series	#benchs	Solver	#series	#benchs
march-eq-100	13	63	sapsnr	11	39	Satzoo_1.02	7	15
satzilla_nr	13	48	satzilla_nr	11	38	march-eq-100	6	27
Satzoo_1.02	12	53	march-eq-100	10	36	brchaff	6	8
brchaff	12	42	Satzoo_1.02	9	38	satzilla_nr	5	10
sapsnr	11	39	Jerusat1.3	9	37	OepirA	5	10
march-007	10	59	march-007	9	36	zchaff	5	9
zchaff	10	41	brchaff	9	34	march-007	4	23
Jerusat1.3	9	45	zchaff	8	32	nanosat	4	8
OepirA	9	41	nanosat	8	31	Jerusat1.3	3	8
nanosat	9	39	OepirA	7	31	sapsnr	0	0
RANDOM								
Solver	#series	#benchs	Solver	#series	#benchs	Solver	#series	#benchs
walksatrnp	27	142	walksatrnp	27	142	kcnfs	4	14
saps	26	144	saps	26	144	march-007	3	9
adaptnovelty	26	143	adaptnovelty	26	143			
GaSAT	26	111	GaSAT	26	111			
satzilla_nr	20	108	satzilla_nr	20	108			
QingTing	19	111	QingTing	19	111			
cls	19	65	cls	19	65			
UnitWalk	17	106	UnitWalk	17	106			
kcnfs	11	60	kcnfs	10	46			
march-007	9	45	march-007	8	36			

Table 1. Results of the first stage for the solvers that entered the second stage by category plus the results of `forklift` and `oepir`

.

For the random category, no solver was able to find a new satisfiable benchmark among those remaining unsolved during the first stage. As a consequence, it was decided with the agreement of the judges, to run the second stage solvers on all the benchmarks they did not solve during the first stage. It can be viewed as re-running all the solvers with an increased timeout on the initial benchmark set. As a consequence, because `adaptnovelty` was able to solve the 150 benchmarks proved satisfiable during the first stage, it was declared winner in the random SAT category. Last year's winner `kncfs` is winning again this year in the UNSAT category. Because of the unbalanced number of proved SAT/UNSAT benchmarks, `adaptnovelty` was also declared winner of the overall random category.

In the crafted category, `march-eq` was awarded in the UNSAT and overall categories while `satzoo-1.02` defended successfully his award in the satisfiable category.

ALL		SAT		UNSAT	
INDUSTRIAL					
Solver	#benchs	Solver	#benchs	Solver	#benchs
zchaff-rand	**20**	**Jerusat1.3**	**3**	**zchaff-rand**	**18**
Satzoo-1.02	10	CirCUsH1	2	Satzoo-1.02	9
brchaff	8	zchaff-rand	2	brchaff	7
Jerusat1.3	6	brchaff	1	quantor	6
quantor	6	compsat	1	CQuest	5
CQuest	5	Satzoo-1.02	1	minilearning	3
CirCUsH1	3	quantor	0	Jerusat1.3	3
minilearning	3	CQuest	0	compsat	2
compsat	3	minilearning	0	CirCUsH1	1
CRAFTED					
Solver	#benchs	Solver	#benchs	Solver	#benchs
march-eq-100	**10**	**Satzoo-1.02**	**2**	**march-eq-100**	**10**
Satzoo_1.02	10	brchaff	1	Satzoo_1.02	8
satzilla_nr	3	nanosat	1	satzilla_nr	3
Jerusat1.3	2	sapsnr	0	Jerusat1.3	2
brchaff	1	satzilla_nr	0	brchaff	0
nanosat	1	march-eq-100	0	nanosat	0
sapsnr	0	Jerusat1.3	0	sapsnr	0
march-007	0	march-007	0	march-007	0
zchaff	0	zchaff	0	zchaff	0

ALL			SAT			UNSAT		
RANDOM								
Solver	#benchs	#new	Solver	#benchs	#new	Solver	#benchs	#new
adaptnovelty	**150**	**7**	**adaptnovelty**	**150**	**7**	**kcnfs**	**26**	**12**
sapsnr	148	4	sapsnr	148	4	march-007	18	9
walksatrnp	145	3	walksatrnp	145	3			
GaSAT	128	17	GaSAT	128	17			
QingTing	113	2	QingTing	113	2			
UnitWalk	112	6	UnitWalk	112	6			
satzilla_nr	110	2	satzilla_nr	110	2			
kcnfs	90	30	cls	78	13			
cls	78	13	kcnfs	64	18			
march-007	63	18	march-007	45	9			

Table 2. Second stage results. On the random category, we also report the number of newly solved benchmarks.

Another interesting data that can be derived from the SAT competition is the smallest instance (with respect to their number of literals) in both SAT and UNSAT categories remained unsolved after the second stage. As stated earlier, those benchmarks should belong to the crafted category, since this is the aim of that category.

The smallest UNSAT benchmark still unsolved after the second stage remains last year's award winner for the smallest unsolved UNSAT benchmark `hgen8-n260-01-S1597732451` (260 variables, 391 clauses, 888 literals) produced by the hgen8 generator[5] submitted last year by Edward A. Hirsch.

Last year's smallest unsolved satisfiable benchmark `hgen2-v400-s161064952` (400 variables, 1400 clauses, 4200 literals) was solved this year by `rsaps` in 461 seconds during the first stage and `sapsnr` in 902s during the second stage.

[5] Available at `http://logic.pdmi.ras.ru/~hirsch/benchmarks/hgen8.html`

The new smallest unsolved satisfiable benchmark was also generated by the hgen2 generator[6] submitted by Edward A. Hirsch for the first competition in 2002 with 450 variables instead of 400:
`hgen2-v450-s41511877.shuffled-as.sat03-1682.cnf` (1575 variables, 450 clauses, 4725 literals).

One can note that the smallest unsolved satisfiable benchmark is significantly larger than the smallest unsolved UNSAT benchmark.

5 State of the Art Contributors

It may be interesting to put all the solvers in a single entity that can decide for each SAT benchmark the best solver to solve it (a sort of perfect `satzilla`). Such entity is called the State-Of-The-Art (SOTA) solver in [19]. Not all the solvers are useful to build the SOTA solver, so we name the solvers that are needed SOTAC (State of the Art Contributor).

Any solver that uniquely solves any benchmark is obviously a SOTAC. Table 3 lists the different solvers that are the only one to solve a given benchmark in the three categories. We computed those numbers from the first stage results, with and without the black boxes `oepir` and `forklift`. The measure can only be made on the first stage results because we need exhaustive results to give all solvers a chance. Thus, some solvers not strong enough to enter the second stage can be distinguished here.

Note that adding `oepirc` and `forklift` does not change much the result: `zchaff` looses 3 benchmarks, `cquest` disappears, `jerusat1.3`, `compsat` and `brchaff` loose one benchmark. Note also that having variants of solvers minimizes the number of uniquely solved instances: with three variants, `oepir` solvers have small numbers of uniquely solved benchmarks compared to the other strong solvers.

In the random category, `kncfs`, `rsaps`, `novely50` and `sapsnr` are three SOTAC in the first stage. Note that `rsaps`, `novely50` and `sapsnr` would not be SOTAC if we take into account the second stage since `adaptnovelty` was able to solve all the SAT benchmarks. It is thus sufficient with a 2400 second timeout to keep adaptnovelty in our SOTA solver to solve all the satisfiable benchmarks solved during the competition in the random category. As a consequence, `adaptnovelty` would be a SOTAC without being the only one to solve one benchmark. Moreover, the smallest SOTA is simply composed by `kncfs` and `adaptnovelty`. Both solvers solves all the solved benchmarks in the random category.

It is more difficult to find the smallest SOTA in the other categories, because one would have to choose the smallest subset of solvers that solves all the benchmarks. If all the SOTAC are indeed in this subset, the choice has to be made for the remaining solvers.

Following the idea of SOTAC, one important issue may be to identify solvers that are subsumed by another solver. Such solvers would be useless in a SOTA

[6] Available at `http://logic.pdmi.ras.ru/~hirsch/benchmarks/hgen2.html`

Without black-boxes				
Solver	Total	Random	Ind.	Crafted
circush0	6	–	–	6 (0/6)
kcnfs	6	5 (0/5)	–	1 (1/0)
jerusat1.3	5	–	4 (2/2)	1 (1/0)
brchaff	4	–	3 (0/3)	1 (1/0)
zchaff	4	–	4 (4/0)	–
quantor	3	–	3 (0/3)	–
satzoo-1.02	3	–	1 (0/1)	2 (0/2)
adaptnovelty	2	–	–	2 (2/0)
circush1	2	–	2 (2/0)	
compsat	2	–	2 (1/1)	–
rsaps	2	1 (1/0)	–	1 (1/0)
zchaff-rand	2	–	2 (0/2)	–
cquest	1	–	1 (0/1)	–
march-eq-100	1	–	–	1 (1/0)
novelty35	1	–	1 (1/0)	–
novelty50	1	1 (1/0)	–	–
sapsnr	1	1 (1/0)	–	–
satzilla	1	–	1 (1/0)	–

Including black-boxes				
Solver	Total	Rand.	Ind.	Craft.
forklift	8	–	8 (1/7)	–
circush0	6	–	–	6 (0/6)
kcnfs	6	5 (0/5)	–	1 (1/0)
jerusat1.3	4	–	3 (1/2)	1 (1/0)
brchaff	3	–	2 (0/2)	1 (1/0)
oepira	3	–	3 (1/2)	–
oepirb	3	–	2 (0/2)	1 (0/1)
oepirc	3	–	3 (0/3)	–
quantor	3	–	3 (0/3)	–
satzoo-1.02	3	–	1 (0/1)	2 (0/2)
adaptnovelty	2	–	–	2 (2/0)
circush1	2	–	2 (2/0)	–
rsaps	2	1 (1/0)	–	1 (1/0)
compsat	1	–	1 (0/1)	–
march-eq-100	1	–	–	1 (1/0)
novelty35	1	–	–	1 (1/0)
novelty50	1	1 (1/0)	–	–
sapsnr	1	1 (1/0)	–	–
satzilla	1	–	1 (1/0)	–
zchaff	1	–	1 (1/0)	–

Table 3. SOTAC ranking: the number indicates the number of benchmarks that sovlers uniquely solve. Numbers in parenthesis detail the number of SAT/UNSAT benchmarks respectively.

solver because all the problems solved by such solver would be also solved by the subsuming solver. However, in some cases, a subsumed solver may be more efficient on its subset of benchmarks. We represent, in figures 2, 3 and 4, the subsumption relations for each category. Dotted lines represent a subsumption relation but with a loss of CPU-time, and plain lines are strong subsumptions, with CPU-time savings. Each edge is labeled with the number of common solved benchmarks. For instance, figure 2 told us that `novely50` subsumes `walksatauto` and `qingting` by solving respectively all the 122 and 111 benchmarks that the solvers also solve. In order to obtain clean figures, we only considered solvers that may solve more than 15 benchmarks in each category and we deleted redundant edges. Thus, a lot of very-simple subsumption relations are missing but are not really interesting ones, especially all the relations with inefficient solvers.

One may observe that while there are many subsumption relations between solvers in the random and crafted category, it is not true in the industrial category. Especially, there is no subsumption relation between the second stage solvers in that category, which is not the case in the random category (`qingting`, `gasat` and `satzilla-nr` are subsumed by `adaptnovelty` for instance) and the crafted category (`nanosat` is subsumed by `march` solvers and `satzoo-1.02`).

6 Other rankings

In order to demonstrate that the competition final results do not change much when the evaluation rules change, we propose here other ways to analyze the results, other methods to rank solvers and to compare the rankings. For instance, in the contest, we tried to forget about the CPU-time, 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 also important to try to characterize efficient solvers, in terms of CPU-time.

6.1 Ranking *à la* SatEx

This ranking used in the SatEx web site[18] allows us to give a picture of solver performances in a very simple but meaningful way. It is not obvious to rank solvers on a basis of a penalty time when the status (SAT or UNSAT) of some benchmarks are not known. We chose to only count, on SAT and UNSAT subcategories, the subset of benchmarks that have been at least solved by a solver (including black-boxes here). Thus, the ranking is preserved even if all or none of the unknown benchmarks are SAT or UNSAT (all solvers will then have to pay the same penalty in the corresponding category).

The winners of the random categories are ranked first using the ranking, same thing for the overall ranking and UNSAT ranking in the industrial category. For the industrial SAT category `compsat` is ranked first but solves fewer benchmarks than `jerusat1.3` and `oepirc`. The same thing happens in the overall and UNSAT crafted categories, where `march-001` is ranked before `march-eq-100`. Concerning the satisfiable crafted category, the incomplete solver `sapsnr` is ranked first closely followed by `satzoo-1.02`. It is the only case for which the SatEx ranking does not guess the second stage winner. The consistency of the SatEx ranking with the contest results also suggests that the time out was large enough to serve as a penalty.

6.2 Relative efficiency of solvers

One of the hard things to handle for ranking solvers is that only partial information is available, because of the timeout. One solution that we adopted last year [12] was to compare only pairs of solvers (X,Y) on the subset of benchmarks they both solve. Let us 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 efficiency* (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. We can compare the relative efficiency of X with respect to 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 *CPU-time reduction ratio* (CRR), and means that, on their common subset of solved benchmarks, X needs $crr(X,Y)$ percent of the time needed by Y. 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 $crr(X)$.

Random benchmarks

All Benchs. (over 300)

Solver	Time (s)	Nb.
adaptnovelty	**97757**	**143**
saps	+747	+1
sapsnr	+1091	0
walksatmp	+1354	-1
novelty35	+4780	-6
satzilla-nr	+18832	-35
qingting	+19141	-32
unitwalk	+20494	-37
gasat	+21722	-32
cls	+51259	-78
kcnfs	+56281	-83
march-007	+60694	-98

SAT (over 150)

Solver	Time (s)	Nb.
adaptnovelty	**7757**	**143**
saps	+747	+1
sapsnr	+1091	0
walksatmp	+1354	-1
novelty35	+4780	-6
satzilla-nr	+18832	-35
qingting	+19141	-32
unitwalk	+20494	-37
gasat	+21722	-32
cls	+51259	-78
kcnfs	+60615	-97
march-007	+63233	-107

UNSAT (over 14)

Solver	Time (s)	Nb.
kcnfs	**4066**	**14**
satzilla_r	+1689	-6
satzilla	+1704	-6
march-007	+1795	-5
march-001	+2064	-7
march-eq-010	+2937	-10
march-eq-100	+4027	-11

Industrial benchmarks

All Benchs. (over 477)

Solver	Time (s)	Nb.
oepirc	**249839**	**95**
forklift	+1751	-14
zchaff-rand	+7842	-28
compsat	+10315	-38
brchaff	+12654	-37
jerusat1.3	+13161	-39
circush1	+16176	-42
cquest	+18532	-50
minilearning.jar	+20066	-51
quantor	+21715	-58

SAT (over 77)

Solver	Time (s)	Nb.
compsat	**28269**	**30**
jerusat1.3	+1506	+3
oepirc	+1715	+4
forklift	+4771	-6
zchaff-rand	+4775	-9
brchaff	+5011	-7
circush1	+8011	-13
cquest	+8687	-11
minilearning.jar	+8717	-11
satzoo_1.02	+11296	-13
quantor	+11695	-20

UNSAT (over 97)

Solver	Time (s)	Nb.
forklift	**36333**	**52**
oepirc	+1695	+3
zchaff-rand	+6477	-12
brchaff	+11053	-23
circush1	+11575	-22
cquest	+13255	-32
quantor	+13430	-31
compsat	+13726	-31
satzoo-1.02	+14484	-34
minilearning.jar	+14759	-33
jerusat1.3	+15064	-35

Crafted benchmarks

All Benchs. (over 228)

Solver	Time (s)	Nb.
march-007	**104789**	**59**
march-eq-100	+2611	+4
satzoo-1.02	+5188	-6
jerusat1.3	+8841	-14
brchaff	+10941	-17
sapsnr	+11235	-20
zchaff	+11832	-18
satzilla_nr	+13280	-11
nanosat	+14362	-20
oepira	+14583	-18

SAT (over 69)

Solver	Time (s)	Nb.
sapsnr	**20624**	**39**
satzoo-1.02	+472	-1
jerusat1.3	+1472	-2
march-007	+2206	-3
brchaff	+2651	-5
zchaff	+3843	-7
satzilla-nr	+4659	-1
march-eq-100	+4735	-3
nanosat	+5444	-8
oepira	+6903	-8

UNSAT (over 41)

Solver	Time (s)	Nb.
march-007	11159	23
march-eq-100	+82	+4
satzoo-1.02	+6922	-8
jerusat1.3	+9575	-15
oepira	+9886	-13
zchaff	+10194	-14
brchaff	+10495	-15
satzilla-nr	+10827	-13
nanosat	+11124	-15

Table 4. Cumulative CPU-Time Ranking (SatEx style), given with relative values. Only second-stage solvers appear here.

However, to have a relevant measure, one have to restrict the set of considered solvers: an inefficient solver, lucky on one instance, may have a very low *re* value. We thus only consider here the set of solvers that participated in the second stage, in each category. Since the first and second stage were done on the same computers for the Crafted and Industrial categories (on LRI's cluster, while the second stage for the random category ran on beta lab's cluster), we

Random (12 Solvers) Solver	re	crr	Industrial (11 Solvers) Solver	re	crr	Crafted (10 Solvers) Solver	re	crr
adaptnovelty	0.95	0.37	oepirc	0.86	0.74	satzoo-1.02	0.89	1.06
walksatrnp	0.95	0.49	forklift	0.77	0.68	march-eq-100	0.86	1.30
sapsnr	0.95	0.56	zchaff-rand	0.73	0.76	march-007	0.84	1.00
saps	0.95	0.67	brchaff	0.66	0.89	satzilla-nr	0.80	1.55
novelty35	0.93	0.86	compsat	0.64	0.70	nanosat	0.79	1.95
qingting	0.81	2.81	satzoo-1.02	0.64	1.94	jerusat1.3	0.77	1.16
satzilla-nr	0.80	0.93	jerusat1.3	0.63	1.55	oepira	0.74	3.13
unitwalk	0.79	2.49	circush1	0.62	1.05	brchaff	0.73	1.28
gasat	0.78	1.82	cquest	0.61	1.17	zchaff	0.72	1.80
cls	0.50	10.81	quantor	0.58	1.43	sapsnr	0.45	0.27
kcnfs	0.43	280.54	minilearning.jar	0.56	1.48			
march-007	0.33	284.76						

Table 5. Relative Efficiency (*re*) and CPU-Time Reduction Ration (*crr*) values for all solvers that participated the second stage. Values are computed over all the solvers of the category, for all benchmarks (SAT and UNSAT).

use extended results (1st stage + 2nd stage + post-competition runs) for those two categories (the timeout may be considered as 2400 s for all launches). The direct consequence of that choice is to increase the number of commonly solved benchmarks. For the random benchmarks, we restrict ourselves to the first stage results only because the second stage was done on a different cluster of computer for that category. Results are given in Table 5.

For the Random category, one can see that `adaptnovelty` solves 95% of the benchmarks that the other solvers solve too (corresponding to the 150 SAT benchmarks), but in only 37% of their time: `adaptnovelty` is definitively the fastest solver in this category. The *crr* then grows to 49% for `walksatrnp`. There is also a clear partition between complete/incomplete solvers. `kncfs` can solve on average 43% of the benchmarks of the other solvers, but in 280 times their CPU time.

For the Industrial category, `oepirc` exhibits the best relative efficiency, showing a very strong solver. It is interesting to see that `forklift` and `zchaff-rand` are very close from a relative efficiency point of view and that `compsat` exhibits the second best *crr* values for this table, which certainly results from its very good running time in the satisfiable category, even if its relative efficiency drops to 64%.

On the Crafted benchmarks, `satzoo-1.02` has the best *re* value, very close to `march-eq-100`. It is interesting to notice the differences between `march-001` and `march-eq-100` for the *crr*. This is likely due to the additional equivalency reasoning used in the latter. The good runtime of `sapsnr` previously observed on satisfiable crafted benchmarks is denoted here by the best *crr* of the category.

6.3 Clustering of solvers according to their performances

We grouped the solver according to their ability to solve each benchmark. Each solver is represented by a vector of boolean indicating whether or not a given

original			first shuffling			second shuffling		
Solver	#series	#benchs	Solver	#series	#benchs	Solver	#series	#benchs
Forklift	10	31	Forklift	10	24	Forklift	10	26
zchaff_rand	9	23	zchaff_rand	9	21	zchaff_rand	9	23
CirCUsH1	9	22	OepirC	8	28	brchaff	9	22
OepirC	8	38	brchaff	8	19	OepirC	8	29
compsat	8	21	CirCUsH1	8	17	compsat	8	17
Jerusat1.3	8	17	CQuest	8	15	CirCUsH1	8	14
brchaff	8	17	compsat	7	19	Jerusat1.3	7	20
Satzoo_1.02	8	16	quantor	7	14	minilearning.jar	7	14
minilearning.jar	8	16	minilearning.jar	7	14	CQuest	7	14
quantor	8	12	Jerusat1.3	6	19	Satzoo_1.02	6	12
CQuest	7	16	Satzoo_1.02	5	12	quantor	6	11

Table 6. Study of the *lisa* syndrome

benchmark was solved by that particular solver. Then we use a hamming distance between those vectors to group the solvers: solvers with similar behavior (solving the same benchmarks) have a small hamming distance. The clusters are represented with a tree figure 5.

7 Discussion

7.1 The effect of shuffling

Shuffling the industrial benchmarks was initially introduced in the competition rules to prevent any cheating. However, two main problems occurred. First, we noticed that two runs of the same solver on the same –but shuffled differently– benchmark can lead to very different results (see *the lisa syndrome* in [12]). Second, competitors claimed that their solver were behaving much better on the original benchmarks than on the shuffled versions used for the competition. It is not a matter of cheating, but seems to be related to clauses and variables proximity observed in real-world problems encoded into SAT. To observe this phenomenon, three benchmarks were associated to each original industrial benchmark: the original one and two shuffled ones. We study here how solvers behaved on those three benchmarks.

Table 6 details the results of the solvers on the original industrial benchmarks (left column) and the same ones shuffled twice. Roughly all the solvers performed better on the original version of the benchmarks, apart `brchaff`, `jerusat1.3` and `quantor`. While the most robust solver with respect to shuffling is `zchaff-rand`, `forklift`, `circush1` and `oepirc` are the solvers that were the most sensitive to the shuffling.

Table 7 focuses on the five series from IBM. `oepirc` is again quite sensitive to shuffling while the other solvers look relatively robust (the number of benchmarks solved is too small to draw any conclusions).

The effect of shuffling the benchmarks in the industrial category is making the benchmarks harder for most of the solvers. Furthermore, the better the solvers are, the more sensitive to shuffling they are.

original			first shuffling			second shuffling		
Solver	#series	#benchs	Solver	#series	#benchs	Solver	#series	#benchs
OepirC	4	22	OepirC	4	16	OepirC	4	15
zchaff_rand	4	13	brchaff	4	13	zchaff_rand	4	13
Forklift	4	13	compsat	4	12	Forklift	4	13
compsat	4	13	zchaff_rand	4	11	brchaff	4	13
CQuest	4	12	Forklift	4	11	CQuest	4	10
brchaff	4	12	CQuest	4	11	compsat	4	10
CirCUsH1	4	11	quantor	4	10	quantor	4	9
Satzoo_1.02	4	9	minilearning.jar	4	8	minilearning.jar	4	8
minilearning.jar	4	9	CirCUsH1	4	8	CirCUsH1	4	8
Jerusat1.3	4	9	Jerusat1.3	3	10	Jerusat1.3	3	10
quantor	4	8	Satzoo_1.02	2	5	Satzoo_1.02	3	6

Table 7. The lisa syndrome, focusing on IBM benchmarks

7.2 Progress or not?

While the number of solvers has constantly increased since the first competition to reach 55 solvers this year, most of the solvers awarded this year are not new. The very same versions of `satzoo-1.02` and `kncfs` were awarded last year in the same categories. `jerusat1.3` was designed a bit after the first competition and participated in the the second one. `zchaff` was one of the first solvers to be awarded in 2002. `march` solvers also participated from the very beginning of the competition. The multi-strategies solvers `oepir` that demonstrated a great potential also participated in the 2003 competition.

These solvers have been improved since last year, for sure. However, none of the strong solvers is a result from a brand new approach.

The good news of this edition lies around local search solvers: the UBCSAT library demonstrated that it was an efficient platform for building various local search solvers. Such a common platform may help developing new efficient local search algorithms for the next competitions. The original `walksat` with `Rnovelty+` strategy also demonstrated its strength, which may tend to prove that `Rnovelty+` strategy itself may be viewed as the winner for the random category.

7.3 Is the competition relevant or not?

Emmanuel Zarpas, from IBM, one of the industrial users embedding SAT solvers as engines in his tools and contributing many benchmarks to the community, published some results he obtained on his own computers on his benchmarks [7], and concluded that it was not possible to tell that Berkmin561 was better than Zchaff on those benchmarks (while berkmin561 performed better than Zchaff during the SAT 2003 competition). A recent update of the experiment showed the Zchaff II (this year winner) was not really better than Zchaff, and performed even worst when taking into account the total CPU time. The underlying question is

[7] http://www.haifa.il.ibm.com/projects/verification/RB_Homepage/bmcbenchmarks_illustrations.html

whether or not the result of the SAT competition means anything in an industrial setting?

In a technical report[26], Emmanuel Zarpas proposed some guidelines for a good BMC evaluation, among them, "use relevant benchmarks, use relevant timeout".

use relevant benchmarks While the aim of IBM is to find the best solver to solve IBM BMC benchmarks, the aim of the competition is to award a robust solver, that is a solver able to solve a wide range of benchmarks across several different kinds of problems split into categories. This different point of view is alone a reason for the discrepancies observed between the results of the competition and those on the IBM benchmarks. Note that some of the IBM benchmarks were part of the SAT 2003 competition benchmarks and the results on those benchmarks were confirmed.

use relevant timeout In his experiment, Emmanuel Zarpas used a 10000 s timeout while we used 600 s for the first stage and 2400 s for the second stage. For practical reasons, we simply cannot afford spending a 10000 s timeout for the competition with 55 solvers. Figure 6 illustrates the results of the complete solvers on all the benchmarks of the industrial category. This representation allows to check the choice of the CPU timeout and to have clues about solver behavior on all the benchmarks in a given category. The 600 s timeout (dotted line) is the first stage timeout. Only the second stage solvers have an extended timeout of 2400 s. Note that this figure was obtained by launching all the second stage solvers on the benchmarks they did not solve during the first stage, which is not the current 2nd stage setting. In our opinion, the first stage results are confirmed for most solvers with that extended timeout. `satzoo-1.02` is the exception: while being ranked 10th after the 600 s timeout, it would be ranked 4th after 2400 s. So the use of a "small timeout" is reasonable to isolate the best solvers in a given category.

8 The next competitions

Several remarks concerning the competition arose this year, and we think it is time to rethink the competition as a whole. For that reason, we already formed the board of judges for the next competition. It will be composed of Oliver Kullmann, Armin Biere and Allen Van Gelder. We already decided that a special track about certified UNSAT answers will take place during the next competition.

We designed some rules for the first competition in 2002 and tried to follow them during the second and third edition of the competition. The major changes since the beginning were the limitation of the number of variants entering the second stage and the board of judges that appeared in the second edition, and the "anti black box rule" and the solver description booklet that was added this year. Apart from that, the competitions were pretty similar.

One first step was to disallow solvers not fully described -either by a detailed report or by its source code- from the competition this year (so-called back boxes). However, it is always difficult to ask both a solver and a detailed report for the same deadline. It was already difficult to obtain a two-page description for each solver this year after the solvers were submitted. In order to fulfill both the aim to open the competition to as many solvers as possible and to award a good fully described solver, we propose to separate the competition in two steps:

– the first one requires only a 2-page description of the solver to enter the first stage, that is to see a solver running shortly on the competition benchmarks.
– a more detailed report will be required (or the source code of the solver) to participate in the second stage and being awardable.

One of the biggest issues for the next competition is to gather adequate benchmarks for the competition. Some of the benchmarks that remained unsolved during the first competition are still unsolved. It is time to get rid of those benchmarks and to find new fresh benchmarks for the competition (or to use good old ones). Furthermore, the balancing between SAT and UNSAT benchmarks of similar difficulty remains the main problem (c.f. UNSAT benchmarks in the industrial category or SAT benchmarks in the random one). Finally, the idea of a booklet dedicated to the benchmarks is appealing.

Acknowledgments

The authors would like to thank the anonymous reviewers for their help to improve that paper, the three judges, Fahiem Bacchus, Hans Kleine-Büning and João Marques-Silva for their help during the competition, and the "Laboratoire de Recherche en Informatique" (LRI, Orsay, France) and the beta lab (UBC, Vancouver, Canada) for providing us with clusters of machines. At last, they thank all the authors of solvers and benchmarks for their participation. The first author was supported in part by the IUT de Lens, The Région Nord/Pas-de-Calais and the Université d'Artois.

References

1. Armin Biere. Resolve and expand. In *this issue*, 2004.
2. 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.
3. Harold Connamacher. A random constraint satisfaction problem that seems hard for dpll. In *this issue*, 2004.
4. Dave A. D.Tompkins and Holger H. Hoos. Ubcsat: An implementation and experimentation environment for sls algorithms for sat and max-sat. pages 37–46, 2004.
5. 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.

6. Hai Fang and Wheeler Ruml. Complete local search for propositional satisfiability. In *Proceedings of AAAI'04*, 2004.
7. 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.
8. Holger Hoos. On the runtime behavior of stochastic local search algorithms for SAT. In *Proceedings of AAAI'99*, pages 661–666, 1999.
9. HoonSang Jin and Fabio Somenzi. CirCUs: A Hybrid Satisfiability Solver. In *this issue*, 2004.
10. Frédéric Lardeux Jin-Kao Hao and Frédéric Saubion. Evolutionary computing for the satisfiability problem. In *Applications of Evolutionary Computing*, number 2611 in LNCS, pages 259–268, University of Essex, England, UK, 14-16 April 2003.
11. 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.
12. Daniel Le Berre and Laurent Simon. The essentials of the SAT 2003 competition. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT2003)*, number 2919 in Lecture Notes in Computer Science, pages 452–467, 2003.
13. 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*, S. Margherita Ligure - Portofino (Italy), May 2003. See also `http://www.cbl.ncsu.edu/publications/`, and `http://www.cbl.ncsu.edu/OpenExperiments/SAT/` .
14. 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.
15. 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.
16. R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S. - T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. K. Ranjan, S. Sarwary, T. R. Shiple, G. Swamy, and T. Villa. VIS: a system for verification and synthesis. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, volume 1102, pages 428–432, New Brunswick, NJ, USA, / 1996. Springer Verlag.
17. B. Selman, H. A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the 12th National Conference on Artificial Intelligence, AAAI'94*, pages 337–343, 1994.
18. 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.
19. Geoff Sutcliff and Christian Suttner. Evaluating general purpose automated theorem proving systems. *Artificial Intelligence*, 131:39–54, 2001.
20. Allen Van Gelder. Another Look at Graph Coloring via Propositional Satisfiability. In *Proceedings of Computational Symposium on Graph Coloring and Generalizations (COLOR02)*, IThaca, NY, September 2002.

21. Allen Van Gelder and Yumi K. Tsuji. Satisfiability Testing with More Reasoning and Less Guessing. In D. S. Johnson and M. A. Trick, editors, *Second DIMACS implementation challenge : cliques, coloring and satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 559–586. American Mathematical Society, 1996.
22. M.N. Velev. Automatic abstraction of equations in a logic of equality. In M.C. Mayer and F. Pirri, editors, *Proceedings of Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX '03)*, number 2796 in LNAI, pages 196–213. Springer-Verlag, September 2003.
23. M.N. Velev and R.E. Bryant. Effective use of boolean satisfiability procedures in the formal verification of superscalar and vliw microprocessors. In *Proceedings of the 38th Design Automation Conference (DAC '01)*, pages 226–231, June 2001.
24. K. Xu and W. Li. Many hard examples in exact phase transitions with application to generating hard satisfiable instances. Technical report, CoRR Report cs.CC/0302001, 2003. http://www.nlsde.buaa.edu.cn/ kexu/benchmarks/benchmarks.htm.
25. Emmanuel Zarpas. Bmc benchmark illustrations. http://www.haifa.il.ibm.com/-projects/verification/RB_Homepage/bmcbenchmarks.html.
26. Emmanuel Zarpas. Benchmarking sat solvers for bounded model checking. Technical report, IBM Haifa Research Laboratory, 2004.
27. 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.

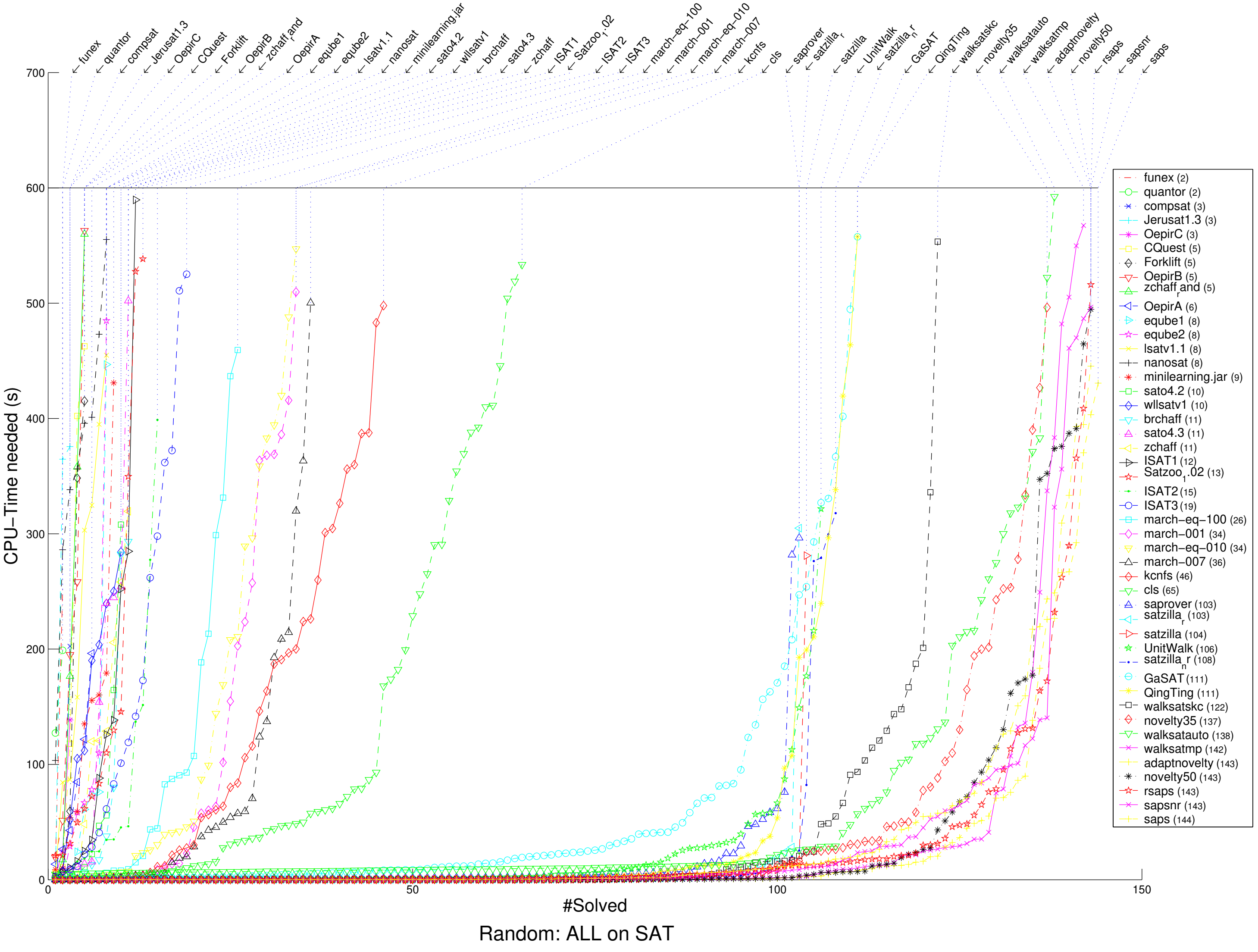

Fig. 1. # of instances solved vs. CPU time for all solvers on SAT instances

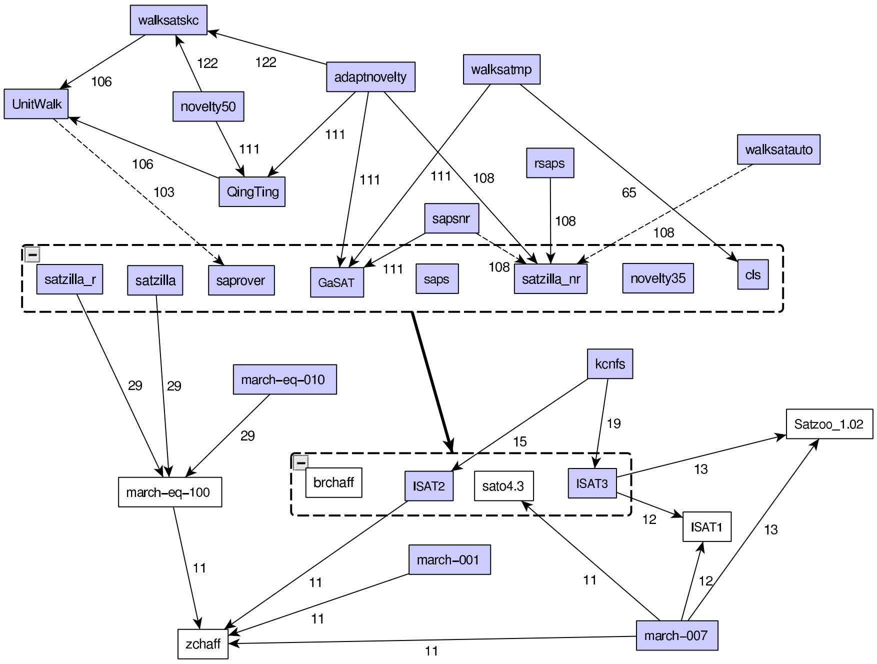

Fig. 2. Subsumptions relations on random instances.

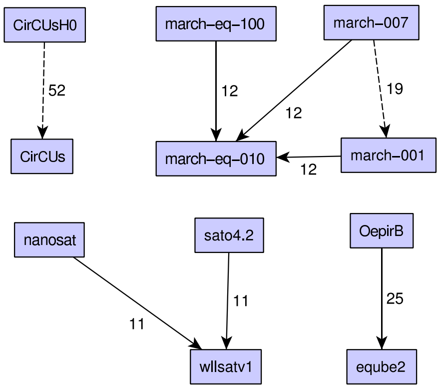

Fig. 3. Subsumptions relations on industrial instances.

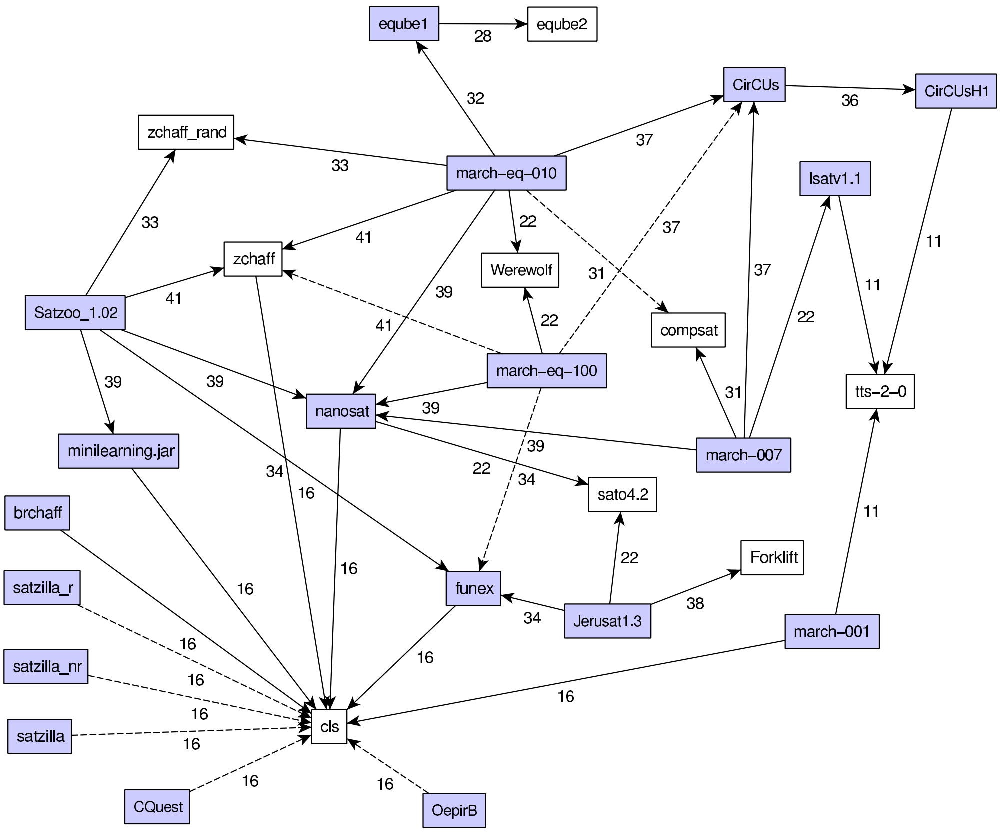

Fig. 4. Subsumptions relations on crafted instances.

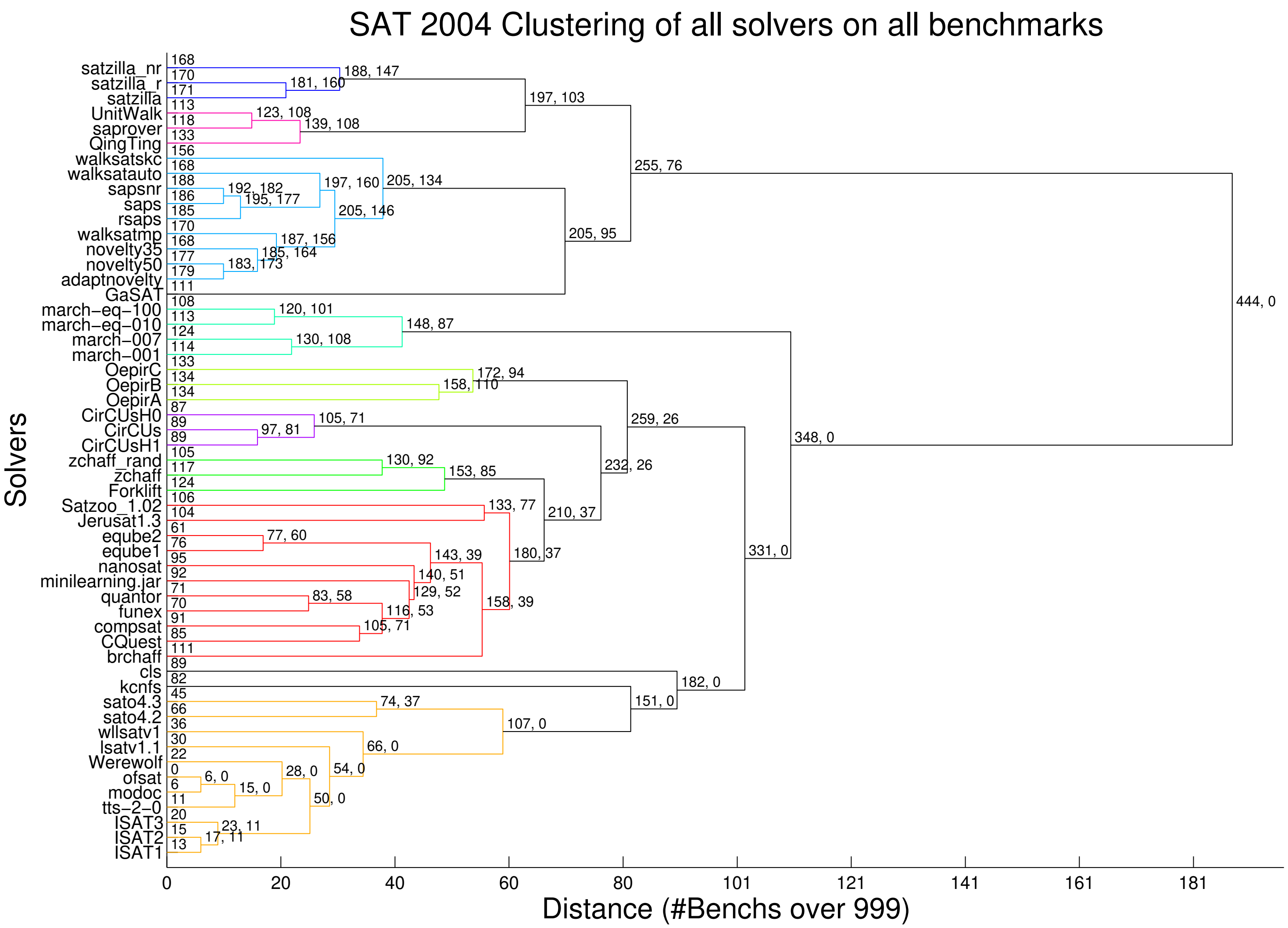

Fig. 5. Clusters of all solvers on all benchmarks. Solvers that closely solve sets of benchmarks are close together in the cluster. The height of nodes in the tree indicates the average distance of the two clusters at each considered branch of the tree. The number on the left indicates the number of instances solved for each solver, and the couple of numbers at each cluster link indicates respectively the number of common benchmarks solved by at least one member of the cluster and the number of benchmarks solved by all the members of the cluster.

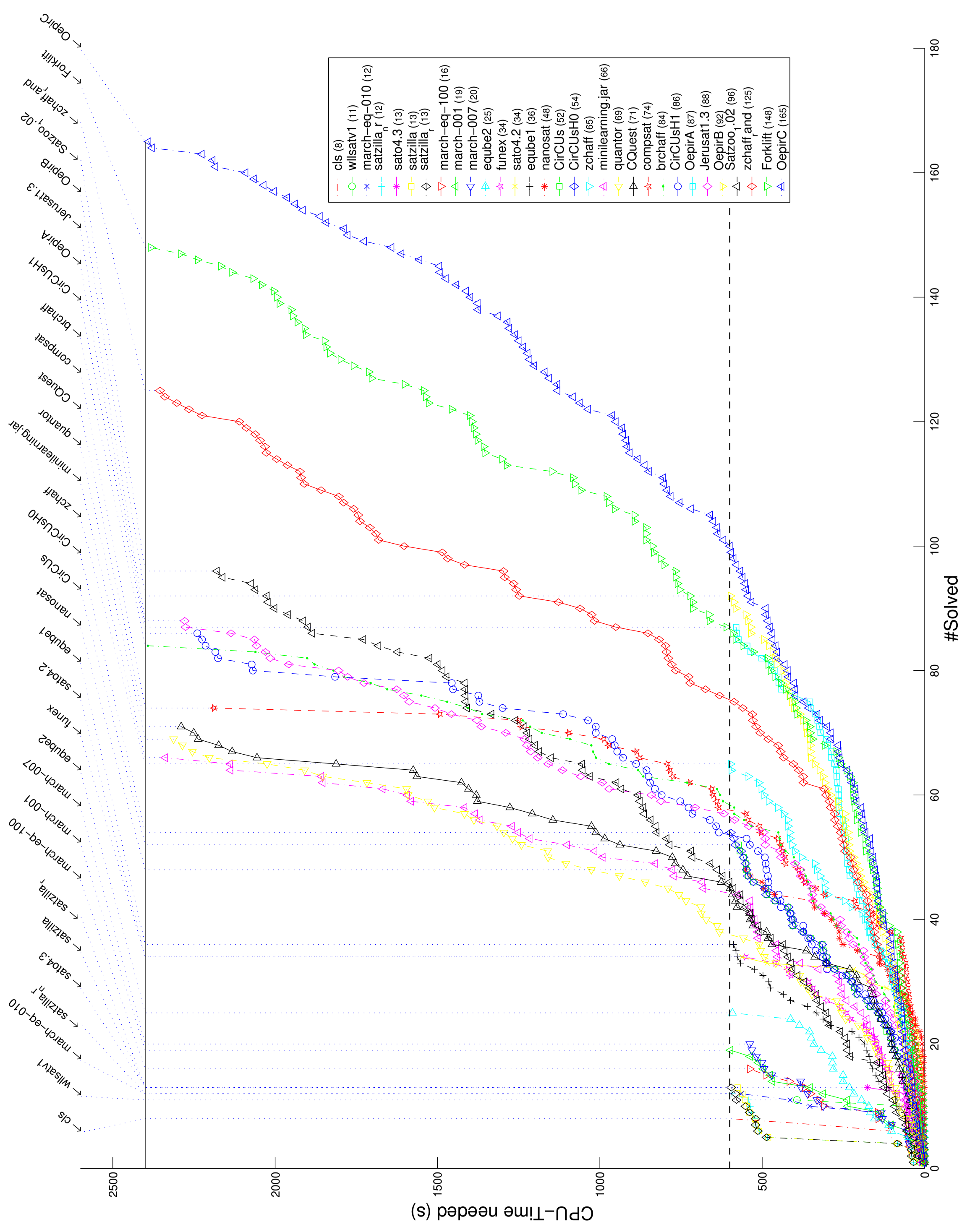

Fig. 6. # of instances solved vs. CPU time for complete solvers on all industrial benchs. Two timeouts were used here, 600s and 2400s, which explain that some curve are stopped before the 600s horizontal limit.