SatEx: A Web-based Framework for SAT Experimentation

Laurent Simon and Philippe Chatalic
Laboratoire de Recherche en Informatique,
U.M.R. CNRS 8623,
Université Paris-Sud,
91405 Orsay Cedex, France
email: {simon,chatalic}@lri.fr

Abstract

SatEx is a web site devoted to SAT experimentation. It is not only a front end to a database gathering an exhaustive number of executions, but it also allows dynamic results synthesis as well as detailed explorations of experimentation results. Being dynamically generated and constantly updated and improved, this site can be considered as an almost always up-to-date SAT experimentation *paper*. To the current time, SatEx presents the results of more than 450 CPU days on a recent machine. In a few months, this site has been well received by the SAT community and has reached more than 20000 hits. SatEx site is available at http://www.lri.fr/~simon/satex/satex.php3.

1 Introduction

While satisfiability testing remains an interesting area for theoretical investigation, most current scientific contributions leave more and more place to experimentation. In particular, one may no more consider presenting any new algorithm without comparing its results with those of other existing algorithms, on a reasonable set of benchmarks. Eight years ago, the Dimacs challenge on satisfiability testing [11] initiated such systematic comparisons by testing an important number of different SAT solvers on different benchmarks. Since that fruitful meeting, interest in SAT solving hasn't stopped growing. Stimulated by this kind of competition, many authors have been constantly improving their algorithms and proposing new benchmark instances, based on real life, or interesting problems, encoded using the SAT formalism [18].

Today, achieving a satisfying experimental comparison is not so easy. While several days where enough at the time of the Dimacs challenge, today, such a systematic comparison would be much more expensive, especially if one wants to compare all systems on the different benchmarks (in CPU-time, it can take more than one year, even on a recent computer). Presenting a new algorithm is therefore a rather tricky exercise. One has to provide the reader with some experimental evidence supporting the good properties of the algorithm but, due to CPU-time limitations, space constraints, etc, one cannot afford an extensive comparison with all other systems on all benchmarks. As a consequence, accounts of experiments are often limited and biased by the experimenter's point of view.

In this paper we investigate on properties that should characterize good SAT experimentations. We first discuss some general principles that most experimental comparisons should satisfy and then focus on the specificity of SAT experimentations. Then we describe SatEx web site, a framework which attempts to address these problems by making available a large database to research community, gathering detailed information on the behavior of most current SAT solvers on most benchmarks. The main motivations for this work are to give the user a precise enough account of experimentation conditions (in order to give him the possibility to recreate the experiment with exactly the same conditions) and to allow him to explore the database by himself and exploits the results without having to run again the same experiments for each couple program/benchmark. Anybody should thus be able to compare his program to others with a minimal effort, just by adding it to the database.

2 Towards fair experimental comparisons of systems

In the following, we assume that the goal is to conduct some experimental comparison of a given system, with other systems able to solve the same task. We further assume that this is performed by running these programs on sets of benchmark instances and that the comparison is performed by analyzing the value of some result parameters. We think that a fair account of such an experimentation should satisfy the following properties:

Reproducibility This is probably one of the most important point. Any user should obtain enough information to be able to obtain again by himself the same results under similar conditions. This requires an absolute *transparency* on the conditions of the experimentation. Characterizing these conditions should be performed at two levels. At the logical level, one should be able to obtain the complete description of the compilation and execution parameters values, used for each run. At the hardware level, one should be able to obtain a precise description of the machine architecture and parameter values. Since machines are improving all the time, one should have some way to compare results obtained on different ma-

chines. Clearly the frequency of the processor has to be taken into account, but it is not sufficient. The amount of memory and also the cache configuration may also have a significant impact on the obtained results. For example, the acceleration induced by cache memory is hard to measure in general and can drastically change results. Given a binary code, we may, on some instances, observe a CPU-time ratio of 10 between two machines, just because of different cache configurations, while on other benchmarks, this ratio may only be of 2 or 3. Therefore, the scale up of CPU-time may be really misleading, and should always be reported when used in a comparison.

Another point of concern is the case of algorithms that integrate some kind of randomness. Reproducibility can simply be obtained by including the random number generation library into the distribution code of the program (thus making the program independent of the operating system, which may include its own random number generation process) and by giving, for each run, the random number seed that deterministically generated the random number series.

Exhaustiveness A satisfying experimentation should also be as *exhaustive* as possible. There may be plenty of benchmarks and it seems hard to restrict the comparison to a subset of them if these choices are not well motivated. Exhaustiveness may be perceived as a real problem because it requires a lot of CPU-time. However, with the relatively low price of computer regarding their performances, it seems difficult today to accept such an argument for restricting the comparison to a subset of benchmarks or programs.

Moreover, experimentation results should neither be forgotten nor deleted. They should remain accessible for verification or further purpose, for example another test series, where previous memorized executions will be naturally omitted.

Objectivity When testing an algorithm, one can hardly think to obtain the best results on all benchmarks (in SAT, this is simply not the case: no algorithm gives the best results on all benchmarks, as pointed out in section 5). Thus, *bad results should also be reported* in any exhaustive and objective account. All experimental sciences are based on bad results as well as good results and it is clear that bad results can be really interesting if they help to understand the behavior of the algorithm being presented [16].

Another point related to objectivity is the choice of the result parameters used for the comparison of algorithms. It is seldom the case that a single parameter is sufficient to obtain a fair comparison. For instance, computing the only sum of CPU-times on a set of benchmark instances doesn't account for the diversity on instances. Other parameters might also be interesting and bring complementary information (average, median, standard deviation,...). Some parameters might be meaningful for some algorithms and not for others. Therefore, the justification of the choice of result parameters used for the comparison should also appear.

To our point of view, any fair experimental evaluation of an algorithm should keep these three criteriae in mind.

3 SAT Experimentations : success and pitfalls

It seems now clear that any claimed advance needs to be supported by some experimental evidence, but, one may wonder whether such experiments are really satisfying and meet the above properties.

Reproducibility is currently rather difficult to achieve. Although most SAT solvers are coded in C and may be recompiled without major difficulty, the precise experimentation conditions are seldom available. Most systems use a number of parameters, the value of which may have a significant impact on performances. Most of the time no hint is given to help adjusting such values. Who has never failed finding again similar results to those reported in a paper?

Exhaustiveness is practically never achieved. There are several explanations for that. One of them is that, although the SAT problem is well defined from a theoretical point of view, it is addressed in different ways by the various SAT solvers. While some of them just try to answer the decision problem, others try to exhibit a model of the formula being tested for satisfiability, and some of them even try to exhibit all models of the formula. Some solvers ensure completeness of the results while others (e.g. incomplete randomized algorithms) don't. This of course has an incidence on the complexity of the underlying algorithms and thus on the kind (and size) of instances that may be solved by each solver. Actually, the existing solvers may be grouped in different families depending on their basic underlying principles (backtrack search, constraint propagation, local search,...) and comparisons are often restricted to the algorithm of a given family.

Benchmarks families used for the comparison may also differ significantly. Instances meaningful for some family might not be interesting for other families. Some benchmarks correspond to models of real world systems, while others have been artificially generated. Some of them have a particular structure (even among those artificially generated) while others have been generated in a uniform random manner. For instance, many work have focused their interest on random generated instances of fixed length clauses (*e.g.* 3-CNF) whose can be easily generated. Moreover, experimental evidence has revealed a particular region of really hard instances [8]. To justify such a restriction, it is often argued that any CNF formula may be turned into some 3-CNF by an appropriate transformation. In addition, [20] adds some evidences that, when most of the structure of a structured instance has been exploited, the resulting formula can be efficiently solved with the same techniques used with random formula.

Today, randomized and incomplete SAT algorithm sometimes present very good experimental studies [17; 26]. Statistical methods have been introduced to report the randomness factor of such experimentation design. In this context, an important effort has been done in the community and some kind of framework for tests that include randomness seems to emerge. But, papers describing complete algorithms tested on non randomly-generated instances are often less cautious. A typical presentation is often organized in two main parts: a

description of the algorithm and a experimentation evidence that it brings new breakthroughs. Even if this is the case and if the experimental part has been carefully performed, this second part is often presented with the aim of increasing the algorithm's value. Typically, only two or three benchmark families are presented, on which the new method performs particularly well. Sometimes, one can also see new arising benchmarks, on which results are impressive but hard to compare: independently of the interest of the benchmarks, if old benchmarks are omitted, it becomes difficult to evaluate the fairness of the comparison. In many cases, good results (which are required to ensure publication) are presented but bad results are omitted.

The lack of exhaustiveness may also result from a lack of CPU-time to perform all possible tests and/or also from a lack of space to report all these results. Such cases generally also lead to some lack of objectivity. Indeed, the choice of a restricted set of systems and instances is inevitably based on the experimenter a-priori point of view on the programs/instances selected for the comparison. Moreover, in most of the cases, results are summarized with a single parameter. Since the details of executions are not available, it is often not possible to consider another criteria for the comparison.

4 SatEx: A framework based on web technologies

In this section, we show that reproducibility, exhausitivity, objectivity and, moreover, incrementality of any experimentation can be reached through an open-web database. The web, associated with the low cost of massive storages allows to keep trace and to dynamically publish all experimentation results, down to the smallest detail. For reproducibility, such a framework guarantees that anybody can read the parameter line and thus run any experiment again in exactly the same conditions. Moreover, the memorization of the program output allows an important saving of CPU-time by publishing exactly what would have been obtained if one had launched the program by its own. This last principle also guarantees the incrementality and the sharing of the database. Thus, if somebody wants to compare a new algorithm, they just have to add it to the base and to check results. Such a web site can be considered as an almost always up-to-date experimentation *paper*. It thus answers the problem of CPU consumption by requiring a minimal cost for any new experimentation (each run is launched only once). Those principles are the main fundations of the SatEx[1] version 2.0, an experimentation framework devoted to SAT which we describe in this section.

4.1 SatEx Overview

Exhaustiveness is a strong point of SatEx. Today, the 2.0 version gathers experimental results corresponding to more than 450 CPU-days[2] on a recent computer (the maximal time given for each execution is 10000s). Roughly speaking, it contains more than 29000 execution traces in a 180 Mb database. SatEx allows to investigate the results of the following 22 SAT provers on 1204 benchmarks:

- For DLL [24] ones, `asat` and `csat` [12], `eqsatz` [22], `nsat` and `sat-grasp` [25], `posit` [13], `relsat` [1], `sato` and `sato-3.2.1` [32], `satz` [23], `satz-213` [21] and `satz-215` [20] and `zchaff` [27].
- For DP [9] ones, `calcres` [4], `dr` [10] and `zres` [6; 5].
- For *randomized* DLL, `ntab` (with `ntab-back` and `ntab-back2`) [8] and `relsat-200` [1].
- And for other approaches, `heerhugo` [15] and `modoc` [29].

From the very beginning, the source code of any of those solvers added to SatEx is verified and, if necessary, modified in order to delete system-dependent random number generation (using a classical random number library), as suggested in section 2. Now, from a benchmark point of view, the database currently contains 1204 benchmarks grouped in the following families: `aim`, `ais`, `bf`, `blockworld`, `dubois`,`f`, `g`, `hole`, `ii`, `jnh`, `parity`, `pret`, `sat-plan` and `ssa` from [11; 18; 28; 19]; `fvp` from [30]; `ucsc-bf` and `ucsc-ssa` from [28; 18]; `beijing` from [7; 18]; `quasigroup` from [32; 18]; `sw100-8-0` and `sw100-8-1` from [14; 18]; `barrel`, `queueinvar` and `longmult` from [3]; `pader-easy` and `pader-hard` from [31] and `joao`, generated from the original `Miters`.

4.2 SatEx: A Guided Tour

The main page of SatEx, a part of which is given figure 1, is splitted in three main parts. The first one presents a summary of the current experimentations status, similar to the previous overview section and, above the top-10 solver part (see section 5.2), an index section allows to jump to the *frequently asked questions* page, the history and the links pages, to download all results (CPU-time and answer of each couple of program/benchmark) and to go to the main experimentation part of SatEx.

This last part proposes an automatic synthesis of results. In the first version of SatEx, only sums of CPU-time over families of benchmarks were given (time needed for a program to finish all the benchmarks of a family, given a maximal time). In the current version, the synthesis includes some refinements such as the average, the standard deviation, the median and a 50% interval confidence. Figure 2 gives an example of a specific synthesis asked by a user. On this example, the user chosen to view all statistics of the executions of 4 solvers and 5 families of benchmarks. It is shown that, over this selection, `posit` is the best on `hfo6` and `hfo6l`, `sato-3.2.1` on `ii-32` and `morphed-8-0`, `sat-grasp` on `Joao` while `satz-215` is the best on the sum of the whole CPU-time.

Now, from this synthesis page, two much more specific synthesis are possible. For example, the user can choose to view the details of all programs behaviors over only one family (by a click on the magnify glass of the desired row). We give in figure 3 the page proposed by SatEx for this kind of

[1] Available at http://www.lri.fr/~simon/satex/satex.php3

[2] On the Dimacs [11] machine scale benchmark, our Pentium-II 400MHz under Linux, taken as a reference in the current SatEx release, has a user time saving of 305%.

Quick overview

The SAT–Ex site attempts to group experimentations around the fundamental SAT problem. Its aim is to gather experiments, providing **executions traces of provers on all benchmarks**. It also provides some study on benchmarks and on programs performances. More information about the SatEx can be found here.

Today, the SAT database can be summarized with the following numbers:

- Total CPU time of the SAT executions databases: **582 days, 20 hours, 53 minutes and 20 seconds** (*of a PII–400 Under Linux*).
- Number of Benchmarks: **1204**, grouped in the following families (beijing, bmc, dimacs, joao, morphed, others, paderborn, quasigroup, ucsc).
- Number of SAT–provers: **23** (asat, calcres, csat, dr, eqsatz, heerhugo, modoc, modoc–2.0, nsat, ntab, ntab_back, ntab_back2, posit, relsat, relsat–200, sat–grasp, sato, sato–3.2.1, satz, satz–213, satz–215, zchaff, zres).
- Number of executions: **30187**

Coming soon (tests are still running): `modoc-2.0`, the 2000' version of modoc.

The Site Index

- Aims and ideas that justify this site, with some papers dealing with the topic. Why doing such a site?
- Frequently Asked Questions and How–To, thus answering several very natural questions about how to navigate in the site, about its construction, or about the experimental conditions and how to gather some results.
- **View synthesis of results** of:
  - Any combination of Solver/Benchmark , where you can choose exactly what solvers and what families of benchmarks should appear in the summary.
  - One of the predefined summaries of results, where you choose only a family of solvers and some origins of benchmarks. Summary tables are cached for speed purpose.
- Download all cpu–time results in CSV–format (gzipped, about 125 Kb);
- View challenging benchmarks.
- The History, with the (too long) list of things to do;
- The traditional Links section.

Top 10 Sat–provers

This rank list is established over all benchmarks of SatEx where results are availables, but this ranking should be taken with care. It is only indicative and it can give a bad or a wrong idea of some program performances, especially for those which have been designed for only some specific benchmarks. A special (and much more complete) TOP page is Here.

Program	Time Used	Slow Ratio	#Solved	#Tested
zchaff	2 days, 16 h. 28 m. 11 s.	1.00	1183	1204
relsat–200	4 days, 7 h. 17 m. 36 s.	1.60	1173	1204
relsat	5 days, 18 h. 18 m. 24 s.	2.15	1158	1204
eqsatz	6 days, 3 h. 12 m. 40 s.	2.28	1162	1204
sato	6 days, 23 h. 20 m. 34 s.	2.60	1157	1204
satz–215	7 days, 8 h. 17 m. 0 s.	2.73	1150	1204
satz–213	7 days, 12 h. 41 m. 39 s.	2.80	1149	1204
sato–3.2.1	8 days, 9 h. 15 m. 50 s.	3.12	1138	1204
satz	8 days, 14 h. 59 m. 6 s.	3.21	1136	1204
modoc	9 days, 20 h. 19 m. 58 s.	3.67	1128	1204

Figure 1: A part of the SatEx main page

User Specific Synthesis (4 solvers, 5 families)

Family \ Program	posit *(v.1.00)*	sat–grasp *(v.2000.00)*	sato–3.2.1 *(v.3.21)*	satz–215 *(v.2.15)*
hfo6 *(40)*	88.28 3.08 2.21 (1.14) [1.06 – 3.17] [0.30 – 3.39] *(40)*	246922.67 10000.00 6173.07 (4613.53) [602.48 – 10000.00] [0.08 – 10000.00] *(17)*	960.57 31.28 24.01 (14.89) [10.73 – 34.69] [0.36 – 49.36] *(40)*	361.37 11.31 9.03 (3.86) [5.49 – 12.08] [2.11 – 13.42] *(40)*
hfo6l *(40)*	2.55 0.07 0.06 (0.01) [0.05 – 0.07] [0.03 – 0.08] *(40)*	76.68 2.77 1.92 (1.58) [0.31 – 3.31] [0.02 – 4.10] *(40)*	5.87 0.18 0.15 (0.07) [0.09 – 0.21] [0.01 – 0.27] *(40)*	41.13 1.04 1.03 (0.04) [1.00 – 1.06] [0.95 – 1.08] *(40)*
ii–32 *(17)*	10099.87 0.51 594.11 (2423.93) [0.17 – 1.48] [0.07 – 10000.00] *(16)*	3.20 0.12 0.19 (0.20) [0.07 – 0.20] [0.03 – 0.79] *(17)*	3.12 0.09 0.18 (0.28) [0.04 – 0.14] [0.01 – 0.90] *(17)*	10219.96 3.39 601.17 (2422.23) [1.40 – 12.70] [0.47 – 10000.00] *(16)*
Joao *(25)*	220130.16 10000.00 8805.21 (3302.24) [10000.00 – 10000.00] [21.52 – 10000.00] *(3)*	182403.59 10000.00 7296.14 (4433.46) [1529.44 – 10000.00] [0.81 – 10000.00] *(7)*	220206.31 10000.00 8808.25 (3293.86) [10000.00 – 10000.00] [1.62 – 10000.00] *(3)*	210452.42 10000.00 8418.10 (3700.01) [10000.00 – 10000.00] [7.70 – 10000.00] *(4)*
morphed–8–0 *(100)*	34.90 0.35 0.35 (0.02) [0.33 – 0.36] [0.29 – 0.42] *(100)*	23.03 0.23 0.23 (0.02) [0.21 – 0.25] [0.18 – 0.29] *(100)*	1.90 0.02 0.02 (0.01) [0.01 – 0.02] [0.00 – 0.03] *(100)*	11.94 0.12 0.12 (0.01) [0.11 – 0.13] [0.10 – 0.14] *(100)*
Total Cpu–Time Required	*2 d 15 h 59 m 15 s*	*4 d 23 h 17 m 9 s*	*2 d 13 h 26 m 17 s*	*2 d 13 h 24 m 46 s*

Figure 2: Example of a user synthesis of families and programs. This array shows a synthesis containing respectively the sum, the median, the mean (with the standard deviation), a 50% interval and the min and max CPU-time values for each cell. Lighter synthesis (displaying only the sum, the mean or the median) are also available.

Family \ Program	csat *(v. 1.00)*	eqsatz *(v. 1.01)*	relsat *(v. 1.12)*	satz–215 *(v. 2.15)*	zchaff *(v. 2.01)*
c1355–s *(3670c, 1294v)*	10000.00 0	10000.00 0	1039.94 20004	10000.00 0	6.88 16463
c1355 *(3662c, 1294v)*	10000.00 0	10000.00 0	2166.85 63111	10000.00 0	7.97 22201
c1908–s *(5108c, 1919v)*	10000.00 0	1445.90 107792	10000.00 0	10000.00 0	13.36 28713
c1908 *(5096c, 1917v)*	10000.00 0	1917.44 171892	10000.00 0	10000.00 0	11.74 25639
c1908_bug *(5100c, 1919v)*	248.63 53184	2.25 4	10000.00 0	10000.00 0	10.59 22216
c2670–s *(7436c, 2940v)*	10000.00 0	10000.00 0	10000.00 0	10000.00 0	6.71 34563
c2670 *(6756c, 2703v)*	10000.00 0	10000.00 0	10000.00 0	10000.00 0	6.15 40217
c2670_bug *(6696c, 2708v)*	10000.00 0	10000.00 0	2.13 216	413.38 0	0.10 216
c3540–s *(9526c, 3499v)*	10000.00 0	10000.00 0	10000.00 0	10000.00 0	202.44 131742
c3540 *(9326c, 3450v)*	10000.00 0	10000.00 0	10000.00 0	10000.00 0	130.96 105305
c3540_bug *(9316c, 3446v)*	208.39 12423	2083.88 61516	5.04 65	10000.00 0	0.14 50
c432–s *(1128c, 392v)*	136.98 390196	31.38 50438	6.90 565	17.61 0	0.26 2327

Figure 3: Details for a family (`Joao` in the example), given a set of programs (`csat`, `eqsatz`, `relsat`, `satz-215` and `zchaff`)

Details on all Execution of a Family

Program : eqsatz *(v. 1.01)*
Launch command : **eqsatz FILE–NAME**
Problem : Parity–32
Number of Instances : **10**
Problem Family : **parity**
Problem Origine : dimacs

	Benchmark	Time	SysTime	Sat	Nb. Nodes	NbV	Nbc
	par32–1	883.75	1.11	SAT	3089	3176	10277
	par32–1–c	1346.50	1.18	SAT	3672	1315	5254
	par32–2	224.11	0.15	SAT	651	3176	10253
	par32–2–c	48.29	0.30	SAT	209	1303	5206
	par32–3	9778.60	7.66	SAT	23827	3176	10297
	par32–3–c	5034.14	3.36	SAT	15123	1325	5294
	par32–4	822.23	1.29	SAT	2885	3176	10313
	par32–4–c	792.09	0.86	SAT	1488	1333	5326
	par32–5	10000.00	7.70	?	––	3176	10325
	par32–5–c	10000.00	7.96	?	––	1339	5350

Figure 4: Details for a family given one program (`eqsatz` on `Parity-32` in the example)

Execution Details

Program : sato–3.2.1 (3.21)
Benchmark : aim–50–1_6–no–4
Launch : **sato–3.2.1 aim–50–1_6–no–4.cnf**
Time : **0.01**
Nodes Number : **6**
Begin date : **2000–08–24 17:07:11**
End date : **2000–08–24 17:07:11**

System Time : **0.00**
Instance From : aim–50
Family : **aim**
Origine : dimacs
Satisfiability : **UNSAT**
Variables : **50**
Clauses : **80**

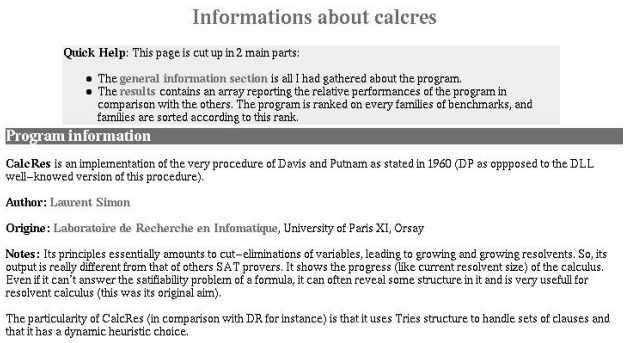
Download this Trace File

Trace File:

```
~~ LAUNCH ON iasi-linux THE 2000-08-24 17:07:11
~~ satexserv 1.0 INTERNAL MARKUPS: 6911 1 14 21 0
~~ REAL COMMAND: /users/iasi/simon/bin/sato-3.2.1 /users/iasi/simon/These/tests/b
~~
------- SATO 3.2.1, 04/2000 on iasi-linux ------
The job "/users/iasi/simon/bin/sato-3.2.1 /users/iasi/simon/These/tests/benchs/D:

c Input file "/users/iasi/simon/These/tests/benchs/Dimacs/aim-50/aim-50-1_6-no-4
```

Figure 5: Publication of the standard output with running statistics for one execution (`sato-3.2.1` on `aim-50-1_6-no-4`). This figure is cutted for lack of place: the original trace file is larger.



Informations about calcres

Quick Help: This page is cut up in 2 main parts:

- The general information section is all I had gathered about the program.
- The results contains an array reporting the relative performances of the program in comparison with the others. The program is ranked on every families of benchmarks, and families are sorted according to this rank.

Program information

CalcRes is an implementation of the very procedure of Davis and Putnam as stated in 1960 (DP as oppposed to the DLL well–knowed version of this procedure).

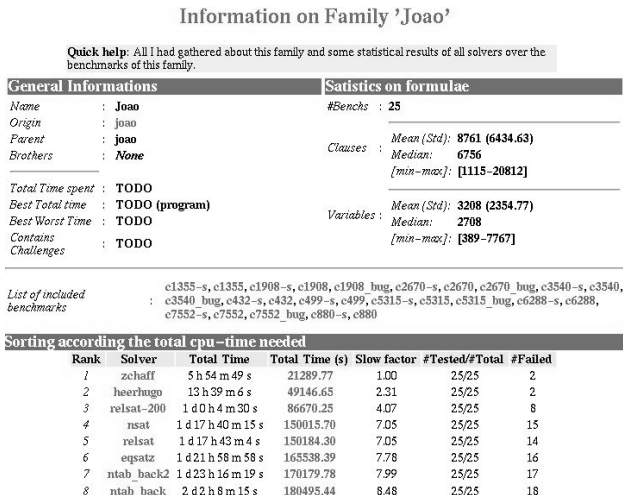
Author: Laurent Simon

Origine: Laboratoire de Recherche en Infomatique, University of Paris XI, Orsay

Notes: Its principles essentially amounts to cut–eliminations of variables, leading to growing and growing resolvents. So, its output is really different from that of others SAT provers. It shows the progress (like current resolvent size) of the calculus. Even if it can't answer the satifiability problem of a formula, it can often reveal some structure in it and is very usefull for resolvent calculus (this was its original aim).

The particularity of CalcRes (in comparison with DR for instance) is that it uses Tries structure to handle sets of clauses and that it has a dynamic heuristic choice.

Figure 6: Information page about a given solver. An array of its results for each families follows.



Information on Family 'Joao'

Quick help: All I had gathered about this family and some statistical results of all solvers over the benchmarks of this family.

General Informations

Name	:	**Joao**
Origin	:	joao
Parent	:	**joao**
Brothers	:	***None***
Total Time spent	:	**TODO**
Best Total time	:	**TODO (program)**
Best Worst Time	:	**TODO**
Contains Challenges	:	**TODO**

Satistics on formulae

#Benchs	:		**25**
Clauses	:	*Mean (Std):*	**8761 (6434.63)**
		Median:	**6756**
		[min–max]:	**[1115–20812]**
Variables	:	*Mean (Std):*	**3208 (2354.77)**
		Median:	**2708**
		[min–max]:	**[389–7767]**

List of included benchmarks : c1355–s, c1355, c1908–s, c1908, c1908_bug, c2670–s, c2670, c2670_bug, c3540–s, c3540, c3540_bug, c432–s, c432, c499–s, c499, c5315–s, c5315, c5315_bug, c6288–s, c6288, c7552–s, c7552, c7552_bug, c880–s, c880

Sorting according the total cpu–time needed

Rank	Solver	Total Time	Total Time (s)	Slow factor	#Tested/#Total	#Failed
1	zchaff	5 h 54 m 49 s	21289.77	1.00	25/25	2
2	heerhugo	13 h 39 m 6 s	49146.65	2.31	25/25	2
3	relsat–200	1 d 0 h 4 m 30 s	86670.25	4.07	25/25	8
4	nsat	1 d 17 h 40 m 15 s	150015.70	7.05	25/25	15
5	relsat	1 d 17 h 43 m 4 s	150184.30	7.05	25/25	14
6	eqsatz	1 d 21 h 58 m 58 s	165538.39	7.78	25/25	16
7	ntab_back2	1 d 23 h 16 m 19 s	170179.78	7.99	25/25	17
8	ntab_back	2 d 2 h 8 m 15 s	180495.44	8.48	25/25	18

Figure 7: Current information page for a family (`Joao` in the example). Two more arrays are below this page: a sorting according to the mean and another according to the median.

synthesis, on which each row corresponds to a benchmark of the given family, and on which results of solvers are symbolized by cell colors (this facilitate bugs revealing). In addition to the CPU-time, the number of explored nodes for DLL algorithms are also given. The last synthesis (see figure 4 for an example) focuses on only one couple of program and benchmark family. Now, from the two previous synthesis, one can access to the detail of an execution (standard output, launching date and running statistics, as given figure 5) by a click on one cell of the first table (of figure 3) or by selecting the magnify glass of the last table (of figure 4). By reading this kind of detail, anybody can check execution conditions and results. The availability of trace files guarantees that, if the automatic synthesis of CPU-time proposed by SatEx is not satisfying, any other measurement can be later computed just by reading all memorized outputs, without doing the whole experiment again. With exhaustiveness, this point clearly answers the problem of fairness and objectivity.

The last part of the experimentation regards the publication of manual and automatic informations about solvers and benchmarks. Thus, each solver has its own page with some informations (when available) and with a summary of its results over all benchmarks, relative to other solvers. An example of the head of such a page is given figure 6. Informations about families of benchmarks are also published (see figure 7). Each page presents three ranking of all solvers over the considered family: one based on the sum of CPU-time, one on the mean and the last on the median CPU-time needed. SatEx also allows comments to be added to each program (in which compilation options or tricks can also appear) as well as to each benchmark family. This last point relies on the emergence of new techniques for SAT. It is clear that current algorithms are so well known and inherits a so important knowledge about optimization that it seems hard to instantly obtain better results with a new but not yet optimized method. Nevertheless, we must give new methods a chance to emerge, and SatEx was also designed for this purpose. The information sections allows global comments on solver behavior to be added, and thus should allow fair comparisons between systems, independently of their optimizations maturity level.

4.3 Internal Structure of SatEx

The version 2.0 of SatEx inherits all the functionalities of the first version and adds a number of new functions in the web page generation process, in the database structure and in the job submission process. Among the possibilities of SatEx 2.0, one can automatically find some bugs of programs (by automatically checking the consistency of program results with respect to the majority of answers) or find up-to-date challenging benchmarks (benchmarks not yet solved by any other programs), as reported section 5.1. In order to facilitate the comparison of programs, SatEx also allows to sort algorithms according to their performances on each families of benchmarks (based on the sum, mean or median value), as reported for example section 5.2.

From an internal point of view, the main evolution of SatEx 2.0 has been to design a special job server that dispatches jobs over a network of 20 biprocessor Linux workstations. This upgrade has been necessary in order to face the incredible amount of CPU-ressource required to maintain such a site. Closely related to this job server, we can find the first of the three databases which make up the heart of SatEx. This database is only devoted to job submissions and output gathering over the network. To guarantee an easy program or benchmark addition, a lot of shell scripts and C programs have been designed. For example, one can add a program just by giving information about it (how to launch it, program family, ...). Benchmarks addition are also easily done just by adding an entry with the path where CNF file are stored: informations about them are automatically computed into the database. In order to manage the queue of jobs to be submitted, one can easily add any couple of program/benchmark just by specifying their characteristics (programs and benchmarks characteristics in the database, such like family or origin names, addition date, ...). In addition, any couple in the queue can be set *resigned* to prevent useless launchs (its CPU-time is then assumed to be 10000s).

When a new release of the SatEx is considered, for instance when a new solver has been added and all tests have been completed, this first database is compiled into a new

one. During this stage, cache tables are computed in order to reduce the web server usage. Program ranking tables, statistics for each couple of program and families (mean, standard deviation, median, max, min and 50% interval) and a lot of other small cache tables (such like the statistics appearing the main page or the list of challenging benchmarks) are computed. This second database is connected to a local intranet web server to test the new release in real conditions. At last, this database is copied to the real web server and the new release is officially published. This last separation was also needed for obvious security reasons.

5 SatEx: a Snapshot

We give here a taste of what a user can learn by visiting SatEx. This section can be viewed as a SAT experimental state-of-the-art report, focusing only on solvers currently reported in SatEx. Of course, the reader is strongly encouraged to visit SatEx site to see an up-to-date version of this report. Note that the current version reports some buggy results from some solvers on particular instances, mostly due to long formulae parsing problems (our versions of `satz`, `ntab`, `ntab-back`, `ntab-back2`, `modoc` and `nsat` are concerned by such small problems on really rare benchmarks). This kind of problem mustn't be considered as an important one: a simple modification in the solver can often fix it. Anyway, at the publication time, SatEx results contains some bugs of which the reader of this report should be aware (we are currently setting all buggies answers to the *resigned* status and, to the best of our knowledge, those bugs don't change our current report).

From the very beginning of SatEx, we wanted to present SatEx results following [32], where results were given with a sum of CPU- time over each families[3]. We give on figures 9, 10 and 11 (at the end of this paper) such a synthesis for all solvers and all families in the SatEx. Best results for a given row is written in bold. As we can see, this kind of report already needs three pages and one can hardly think reporting much more informations in such a hardcopy synthesis, which strongly encourage our web publication idea underlying SatEx.

We will not fully comment these tables here. Nevertheless, one remark that can be drawn from these tables is that there is no algorithm surpassing all others on all benchmarks. For instance, `zchaff`, which gives really impressive results (see section 5.2), fails on some hard instances, even on structural ones (`barrel` or `hfo6` for example).

5.1 Challenging benchmarks

One of the new functionalities of SatEx 2.0 is to propose a challenging benchmark section. A challenge is here defined as a benchmark not yet solved by any program during the given time. This part of SatEx allows one to easily find up-to-date challenges, which is often a hard task today. We show that, surprisingly, over the 1204 considered benchmarks, only 8 of them are challenge ones, and 2 of them are known to be solved by `eqsatz` in a larger amount of time:

- `f1000`, `f2000`, from the `f` family.
- `g125.17`, `g125.18`, `g250.15`, `g250.29`, from the `g` family.
- `par32-5-c`, `par32-5`, from the `Parity-32` family.

In addition, SatEx provides a list of benchmarks solved by only one or two provers. When we read the following list, it is striking to remark the number of different program that solves such virtually challenging benchmarks. Today, these 16 benchmarks are:

- `2dlx_ca_mc_ex_bp_f`, `2dlx_cc_mc_ex_bp_f`, `9vliw_bp_mc`, from the `fvp` family (solved by `zchaff`).
- `3bitadd_31`, from the `Beijing` family (solved by `satz-215`, which is `satz-214` (not available in SatEx) with detection of implied literals [20]).
- `c3540-s`, `c6288-s`, `c6288`, from the `Joao` family (solved by `zchaff` and `heerhugo`).
- `f600`, from the `f` family (solved by `satz`).
- `par32-1-c`, `par32-1`, `par32-2-c`, `par32-2`, `par32-3-c`, `par32-3`, `par32-4-c`,`par32-4`, from the `Parity-32` family (solved by `eqsatz`).

5.2 An Attempt of Solver Ranking

We give on figure 8 the ranking of solvers proposed by SatEx. Of course, such ranking must be taken with great care. Firstly, it is only indicative and it can give a bad or a wrong idea of some program performances, especially for those which have been designed for only some specific benchmarks. Secondly, this ranking can be easily modified by adding to the SatEx some particular benchmarks (e.g. random uniform formulae, formulae with a lot of equivalency clauses, structured benchmarks...). And, lastly, the CPU cut-off parameter may be also misleading. What happens to this list if we modify this value?

We give on the same figure (fig. 8) a first answer to this question by considering the whole database with two smaller cut off parameters in the 3 last columns of the figure (respectively with values 1000s and 200s instead of 10000s). We compute the same top while discarding results that needed more than 1000 seconds (or respectively 200s, on which we also provide the number of solved instances). One conclusion that may be drawn is that, surprisingly, the cut-off principle is strong enough for this kind of ranking. Besides the case of `sato`, most solvers keeps more or less the same place in the ranking. If this suggests that a satisfying experimentation can be drawn from small cut-off values, it also suggests that considering 10000s is large enough in most cases and should put bad solvers at a disadvantage by bounding their performances with a large CPU-time value. One last remark can be raised from this figure: the robustness of the cut-off value can, in great part, be due to the easiness of benchmarks, most of which can be easily solved by recent and optimized solvers. In the future, some new and much harder benchmarks (for example from VLSI [**?**], where new huge benchmarks are available) will be added and will certainly change the robustness of the cut-off choice.

[3] In such a table, if a solver can't handle a benchmark in less than 10000s, then this cut off parameter is taken as the time needed by the solver (we discuss in section 5.2 this choice)

Program	Total Time	#Solved	#Tested	Rank	Rank 1000s	Rank 200s	#Solved 200s
zchaff	2 d, 16 h, 28 h	1183	1204	1	1	3	1147
relsat-200	4 d, 7 h, 17 h	1173	1204	2	2	1	1149
relsat	5 d, 18 h, 18 h	1158	1204	3	3	2	1137
eqsatz	6 d, 3 h, 12 h	1162	1204	4	5	8	1119
sato	6 d, 23 h, 20 h	1157	1204	5	12	12	1056
satz-215	7 d, 8 h, 17 h	1150	1204	6	4	4	1114
satz-213	7 d, 12 h, 41 h	1149	1204	7	6	6	1108
sato-3.2.1	8 d, 9 h, 15 h	1138	1204	8	8	7	1105
satz	8 d, 14 h, 59 h	1136	1204	9	7	5	1106
modoc	9 d, 20 h, 19 h	1128	1204	10	11	11	1074
ntab-back2	10 d, 9 h, 59 h	1120	1204	11	9	9	1082
ntab-back	11 d, 5 h, 29 h	1111	1204	12	10	10	1083
sat-grasp	19 d, 7 h, 44 h	1055	1204	13	13	13	994
csat	21 d, 18 h, 17 h	1031	1204	14	14	15	967
posit	22 d, 15 h, 45 h	1020	1204	15	15	14	967
nsat	27 d, 11 h, 31 h	988	1204	16	16	16	923
heerhugo	32 d, 5 h, 43 h	946	1204	17	18	18	862
ntab	33 d, 18 h, 15 h	921	1204	18	17	17	885
asat	39 d, 14 h, 14 h	871	1204	19	19	19	823
zres	96 d, 17 h, 10 h	408	1204	20	21	21	99
calcres	99 d, 12 h, 37 h	353	1204	21	20	20	221
dr	131 d, 15 h, 25 h	67	1204	22	22	22	64

Figure 8: Top solvers as given in SatEx and crossing values for different cut-off parameters

More lightly, one can also note that DP-based solvers hold the tail of the list. As two of them were proposed by us, those bad results give us a kind of authorization (at least a moral one) for the comparison and the ranking of other works. We hold ourselves our own last places.

6 Related and Future Work

Historically, previous web-based work were proposed before SatEx. One of the most important one is SatLib [18]. This web site proposes to distribute benchmarks and solvers. Today, this site is simply the best reference of the topic. The strong point of SatLib is its benchmark distribution section and its important number of pages detailing each benchmarks families proposed for downloading – most of benchmarks included in SatEx are indeed from SatLib –. Unfortunately, the solver section is not so complete and a lot of programs in SatEx were directly requested from their author or their web site. One of the weakness of SatLib is its static aspect, and, in this context, SatEx fully complete SatLib by providing detailed experimental information on all this benchmarks and solvers. As SatLib is already well known and often used by the community, we doesn't plan to distribute benchmarks or solvers directly from SatEx. Another web project, Sat Live! [2], was proposed to facilitate and encourage discussion and diffusion of ideas and papers. One of the policies of Sat Live! is *updating* and, in this sense, it also fully completes the SatLib web site by providing constantly fresh informations about papers and solvers.

Now, from a SatEx point of view, the near future first implies some important benchmark additions. For instance, random formulae are not well represented in the current version and should allow a more satisfying representation of algorithm performances. Comments (including tricks for solvers) should also be added for each solver as suggested in section 4.2. For this purpose, we opened a forum on Sat Live!, showing the complementarity of the two dynamic web-based approaches.

6.1 Scaling-up CPU-time results

In the future, we also plan to add support for evaluating randomized and local search algorithms, for which thousands runs are needed for each benchmark. The special job server was also designed in this aim. But, this imply to scale-up CPU-time from different machines. This is one of the biggest problem facing SatEx evolution. We also find problem again when we want to anticipate and follow computer evolution, which must be taken into account to provide an up-to-date site in even a couple of years. One of the solutions for this can be to launch all previous failed executions again, with the same cut-off parameter on more recent machines and then to automatically upgrade the results of the whole database by scaling-up all its results to a new reference machine. But, scaling-up CPU-time is a hard – and mostly impossible – task. It seems impossible to find a law for scaling results in a fully satisfying way: too many factors play important roles in an execution (Main memory, CPU family, cache memory size and configuration, bus size, operating systems, compilers, ...), and the problem will be even much worse in the future: it is believed for example that no simple law (with less than hundreds factors) exists to transform a CPU-time report from a Pentium III to the fully new Pentium 4 architecture.

At last, the reader should be aware of another CPU-time report problem. For instance, CPU-time measurements are of a limited precision for very quick answers and, even on the same computer, a 20% error is often observed on small CPU-time values (due to small errors in CPU-time operating system measurement). A reported time smaller than a dozen

seconds is clearly imprecise enough to bias a comparison, and especially when this comparison reports the sum of hundreds imprecise small values.

Some partial solutions are indeed possible. First, we can forget our job server, focusing for now on executions on the same machine. When significantly faster machines will be available, we launch the whole experiments again on a network built with only the same machines. If this can be easily done by reusing all the SatEx architecture, this solution is clearly unsatisfying from the incrementality point of view. Another solution can be based on approximation and bounding of the scaling process. This require an important experimental study of programs behavior over the different architecture but should be fully satisfying for providing a good taste of program behavior and comparison. The last and certainly most satisfying answer is to forget architecture problems and to build a new SatEx based on a kind of virtual machine on which C compiler are available, and to base the whole experimentation process on this machine. Cpu-time reports will be then independent of the machine running the virtual one.

Aknowledgements: Authors would like to thank Daniel Le Berre for fruitful discussion about SatEx from the very beginning, and of course all solvers and benchmarks authors whose works are reported in SatEx.

7 Conclusion

In this paper, we have tried to define properties that characterize good experimentations. We have noticed that almost all current SAT experimentations aren't fully satisfying, although they often rely on their experimental results to validate their new approach. We propose a different approach of experimentation, based on the publication of detailed results as well as an automatic synthesis that uses a dynamic and constantly evolving web site. For instance, all the results of `zchaff`, an efficient new solver, were available only some days after its public release and anybody had the opportunity of comparing its results with previous approaches. In this sense, we think that a tool like SatEx is at least as important for the community as a new scientific result.

We think that, for incrementality, homogeneity, transparency, program/benchmark distribution and program behavior study (which is the basis of experimentation in other areas), a project like SatEx is necessary. Moreover, we think that this framework could also be used in other fields, where benchmarking plays an important role to compare and motivate new approaches. The web technology allow us to publish all the details of all executions that are presented in any paper reporting an experimental analysis. Such a framework will at least guarantee fairness and unbiased results to be published.

	asat	calcres	csat	dr	eqsatz	heerhugo	modoc	nsat
aim-100	4471.54	166839.92	0.90	240000.00	1.73	1.72	0.30	0.18
aim-200	130040.78	240000.00	0.77	240000.00	2.48	4.60	1.20	1.35
aim-50	0.60	80995.24	9.57	172050.33	3.89	1.31	0.12	**0.06**
ais	118.23	30621.24	1.79	40000.00	3.15	2632.66	33.76	3145.74
barrel	11813.07	50650.75	30882.91	70000.08	**13.15**	10047.34	15923.16	80000.00
Beijing	106170.40	160000.00	94820.58	151022.98	66484.11	120084.15	29259.35	91751.40
bf	30102.79	20270.06	29611.84	40000.00	44.81	11.26	5.76	10000.33
blockworld	10071.89	50009.58	134.23	60000.42	6189.93	20061.68	20004.29	19280.21
dubois	54398.16	2.42	100729.21	1.37	0.39	0.96	72479.47	0.44
f	30000.00	30000.00	30000.00	30000.00	30000.00	30000.00	30000.00	30000.00
fvp	40000.00	40000.00	40000.00	40000.00	34954.60	30211.16	40000.00	10000.30
g	40000.00	40000.00	40000.00	40000.00	40000.00	40000.00	40000.00	40000.00
hfo3	128.21	400000.00	33.62	400000.00	387.05	236491.02	12476.28	332088.31
hfo3l	2.13	400000.00	**1.55**	400000.00	10.96	48336.30	23.26	2364.96
hfo4	198.20	400000.00	109.46	400000.00	1093.24	320669.22	1985.01	371181.91
hfo4l	2.43	400000.00	1.92	400000.00	11.53	204891.98	7.96	353.41
hfo5	218.16	400000.00	162.53	400000.00	1312.78	292061.75	1211.70	341623.47
hfo5l	2.46	400000.00	2.44	400000.00	16.33	210290.31	5.70	105.80
hfo6	238.09	400000.00	219.17	400000.00	1130.80	321003.22	924.82	295112.34
hfo6l	3.48	400000.00	4.52	400000.00	43.42	224831.48	7.50	64.75
hole	158.64	40205.66	131.47	40639.76	4467.96	26409.21	283.95	11444.67
ii-16	51176.43	100000.00	50072.16	100000.00	261.18	76976.01	17946.93	56146.66
ii-32	503.60	170000.00	59.63	170000.00	390.58	31048.15	25265.59	20488.35
ii-8	48118.72	130000.08	10924.14	130000.10	11.52	15.59	1.74	0.80
jnh	1.43	500000.00	1.63	500000.00	8.30	929.25	2.76	14.10
Joao	221827.73	250000.00	210754.62	250000.00	165538.39	49146.65	190127.91	150015.70
longmult	29897.40	120100.66	31587.56	140011.19	24329.35	93378.31	125206.79	32071.97
morphed-8-0	8.32	1000000.00	11.47	1000000.00	24.21	123.28	3.84	2.91
morphed-8-1	23934.39	1000000.00	10016.94	1000000.00	23.95	114.15	4.13	3.03
Parity-16	60.53	100000.00	119.91	100000.00	**3.26**	94658.63	490.12	840.61
Parity-32	100000.00	100000.00	100000.00	100000.00	**38929.71**	100000.00	100000.00	100000.00
Parity-8	0.17	13.10	0.68	50002.95	0.53	1.15	0.23	0.18
pret-150	40000.00	40000.00	40000.00	27.71	**0.14**	11.13	40000.00	0.27
pret-60	111.36	0.70	40000.00	0.99	0.14	2.03	202.24	0.09
Quasigroup	120787.26	220000.00	8196.47	220000.00	9544.89	145102.33	41995.12	31520.82
queueinvar	22615.42	80576.83	2797.93	100000.00	55.10	35000.89	23957.10	10571.31
satplan	45718.63	90000.00	6243.13	90000.00	6194.54	20553.68	20781.92	26163.90
ssa	30369.78	3495.33	3614.57	60021.63	6.04	4.92	2.53	20000.59
ucsc-bf	1817410.75	520368.31	921310.88	2060116.50	98378.53	270.74	126.47	183948.38
ucsc-ssa	410186.22	24875.73	77667.95	940034.69	88.30	49.91	49.94	103986.69

Figure 9: Synthesis (sum of cpu time temps CPU) from SatEx, part (A), number of solved instances not reported

	ntab	ntab-back	ntab-back2	posit	relsat	relsat-200	sat-grasp
aim-100	13395.03	1117.15	6.42	310.56	1.46	2.04	0.52
aim-200	74575.71	30213.42	810.12	111169.93	2.28	3.55	8.29
aim-50	0.6	0.57	0.5	0.42	2.72	2.55	0.24
ais	19.02	21.6	31.52	0.71	26.9	44	253.15
barrel	30313.67	30365.52	30392.46	50000.95	32248.14	11872.8	41204.48
Beijing	130022.9	130019.63	111494.68	91038.86	24370.74	24024.57	34978.47
bf	30071.16	10028.3	10008.2	20022.69	6.34	2.18	2.1
blockworld	20000.67	20000.66	20000.67	10015.6	618.9	491.88	1510.41
dubois	55537.98	54682.96	55579.96	63167.4	**0.18**	0.27	9.65
f	30000	30000	30000	30000	30000	30000	30000
fvp	40000	30000.2	30000.19	40000	30022.31	30006.79	30009.3
g	40000	40000	40000	40000	40000	40000	40000
hfo3	118.14	157.49	224.48	47.96	196.93	309.78	168115.47
hfo3l	3.02	3.6	4.84	1.83	5.08	6.39	129.54
hfo4	288.01	356.59	500.45	**91.23**	710.13	569.51	287870.12
hfo4l	4.19	4.9	6.2	**1.56**	7.31	7.56	204.47
hfo5	508.62	579.25	786.75	**112.12**	973.98	561.13	293962.22
hfo5l	5.01	5.59	7.09	**1.78**	13.07	16.18	106.42
hfo6	585.95	622.51	831.14	**88.28**	1059.85	595.52	246922.67
hfo6l	7.08	7.55	9.52	**2.55**	45.33	68.68	76.68
hole	**21.67**	27.92	34.67	279.41	786.58	1113.67	11520.25
ii-16	10726.18	5.72	4.98	20040.67	115.63	62.41	10102.68
ii-32	3.87	3.87	3.71	10099.87	952.69	812.22	3.2
ii-8	1.28	1.4	1.35	5.43	4.05	2.94	5.01
jnh	2.53	2.67	2.72	1.62	11.5	6.93	5.69
Joao	200042.89	180495.44	170179.78	220130.16	150184.3	86670.25	182403.59
longmult	100011.99	90015.54	90014.27	14461.11	80743.97	41243.77	82309.19
morphed-8-0	7.34	8.02	8.34	34.9	7.2	9.15	23.03
morphed-8-1	7.52	7.6	8.38	10521.62	7.81	9.3	22.86
Parity-16	61.22	83.85	106.58	24.45	151.7	117.04	18710.47
Parity-32	100000	100000	100000	100000	100000	100000	100000
Parity-8	0.35	0.35	0.4	0.31	0.46	0.48	0.18
pret-150	40000	40000	40000	40000	0.58	0.4	2.63
pret-60	383.02	558.23	624.74	323.07	0.1	0.08	0.23
Quasigroup	62303.24	62372.76	33777.16	100073.17	2864.29	2347.83	86544.53
queueinvar	22665.81	24694.4	4155.49	51534.56	643.6	236.04	764.23
satplan	50001.11	26047.6	20006.86	20027.71	633.5	492.93	1549.14
ssa	23103.43	10338.2	11631.79	10061.91	8.49	2.52	2.68
ucsc-bf	1668129.12	24039.17	44677.9	871828.06	357.73	117.22	84.18
ucsc-ssa	174027.27	33299.29	54045.06	32022.82	118.79	25.57	25.88

Figure 10: Synthesis (sum of cpu time) from SatEx, part (B), number of solved instances not reported

	sato	sato-3.2.1	satz	satz-213	satz-215	zchaff	zres
aim-100	**0.14**	**0.14**	1.9	1.58	1.91	0.21	160189.27
aim-200	0.45	**0.35**	2.35	2.29	2.55	0.7	217412.23
aim-50	0.08	0.09	7.27	3.59	3.8	0.14	8350.63
ais	**0.24**	0.29	0.53	0.55	0.66	57.92	30013.04
barrel	10417.7	11225.41	24670.02	12575.32	7494.03	912.22	50862.7
Beijing	44078.88	23814.7	70066.74	32086.73	21290.58	**20268.12**	146296.55
bf	1.25	6.23	6.13	6.22	12.59	**0.63**	14008.88
blockworld	244.58	782.48	10003.62	1286.26	1048.68	**41.31**	50009.41
dubois	0.56	0.19	39771.01	54129.45	49967.97	0.2	47.23
f	30000	30000	**25576.86**	30000	30000	30000	30000
fvp	30006.35	40000	36908.74	36927.21	30455.28	**1224.86**	40000
g	40000	40000	40000	40000	40000	40000	40000
hfo3	3479.41	759.99	24.5	**23.57**	26.35	1756.19	400000
hfo3l	7.07	4.57	2	2.55	3.23	5.42	400000
hfo4	33785.92	2390.02	210.84	177.01	182.53	6506.1	400000
hfo4l	8.5	5.68	3.96	4.43	5.18	6.62	400000
hfo5	42986.17	1839.89	288.53	276.48	279.23	3267.82	400000
hfo5l	6.34	5.47	9.96	10.9	11.52	5.32	400000
hfo6	23675.85	960.57	362.57	360.49	361.37	1151.53	400000
hfo6l	5.87	5.87	38.7	40.4	41.13	4.6	400000
hole	180.5	81.64	100.95	129.59	144.2	42.81	21027.13
ii-16	**2.14**	30007.98	49.15	47.99	46.17	69.23	100000
ii-32	5.19	**3.12**	10213.66	10225.97	10219.96	3.41	170000
ii-8	**0.43**	0.44	2.41	2.7	2.77	0.79	120151.65
jnh	0.99	**0.86**	6.65	7.7	8.15	1.46	500000
Joao	199291.88	220206.31	190332.83	210349.48	210452.42	**21289.77**	201957.88
longmult	22270.98	7590.25	46541.15	38165.44	42955.44	**4502.49**	120992.63
morphed-8-0	1.99	**1.9**	8.85	10.24	11.94	3.83	1000000
morphed-8-1	**1.83**	2.01	543.05	12027.97	12102.34	3.95	1000000
Parity-16	152.81	230.88	70.16	56.84	62.37	29	98486.32
Parity-32	100000	100000	100000	100000	100000	100000	100000
Parity-8	0.1	**0.08**	0.43	0.48	0.59	0.14	189.39
pret-150	1.19	933.53	40000	40000	40000	0.15	23.97
pret-60	0.11	0.13	215.62	404.94	143.38	**0.04**	3.82
Quasigroup	1087.7	**337.61**	50361.22	1049.75	986.16	845.89	220000
queueinvar	20400.45	21586.18	21133.79	1458.83	10040.4	**15.39**	61344.6
satplan	242.83	1111.79	10069.59	1226.86	1069.87	**43.17**	90000
ssa	1.59	8508.05	425.64	427.04	273.77	**0.89**	11824.02
ucsc-bf	63.83	84620.58	9750.18	9922.81	6324.15	**22.28**	490168.97
ucsc-ssa	22.62	97525.28	17364.5	17069.68	18587.83	**6.83**	62866.28

Figure 11: Synthesis (sum of cpu time temps CPU) from SatEx, part (C), number of solved instances not reported

References

[1] R.J. Bayardo and R. Schrag. Using CSP look-back techniques to solve real-world sat instances. In *14th National Conference on Artificial Intelligence*, pages 203–208, 1997.

[2] Daniel Le Berre. The Sat Live! web site. http://www.satlive.org.

[3] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *Proceedings of Tools and Algorithms for the Analysis and Construction of Systems (TACAS'99), number 1579 in LNCS*, 1999. Benchmarks available at http://www.cs.cmu.edu/~modelcheck/bmc.

[4] Ph. Chatalic and L. Simon. Davis and Putnam 40 years later: a first experimentation. Technical Report 1237, LRI, Orsay, France, 2000.

[5] Ph. Chatalic and L. Simon. Multi-resolution on compressed sets of clauses. In *12th International Conference on Tools with Artificial Intelligence, ICTAI-2000*, pages 2–10, 2000.

[6] Ph. Chatalic and L. Simon. Zres: the old DP meets ZBDDs. In *Proceedings of the 17th Conference of Automated Deduction (CADE)*, pages 449–454, 2000.

[7] James Crawford. International competition and symposium on satisfiability testing. March 1996.

[8] J.M. Crawford and L.D. Auton. Experimental results on the crossover point in random 3sat. *Artificial Intelligence*, 81, 1996.

[9] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, pages 201–215, 1960.

[10] R. Dechter and I. Rish. Resolution versus search: Two strategies for sat. *Journal of Automated Reasoning*, 24(1/2):225–275, February 2000.

[11] The DIMACS challenge benchmarks. from the site ftp://ftp.rutgers.dimacs.edu/challenges/sat.

[12] O. Dubois, P. André, Y. Boufkhad, and J. Carlier. Sat versus unsat. In *Dimacs challenge on Satisfiability Testing*, 1993.

[13] Jon William Freeman. *Improvements to propositional satisfiability search algorithms*. PhD thesis, University of Pennsylvania, 1995.

[14] Ian P. Gent, Holger H. Hoos, Patrick Prosser, and Toby Walsh. Morphing: Combining structure and randomness. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99)*, pages 654–660, Orlando, Florida, 1999.

[15] Jan Friso Groote and Joost P. Warners. The propositional formula checker heerhugo. *Journal of Automated Reasoning*, 24(1/2):101–125.

[16] J. N. Hooker. Needed: An empirical science of algorithms. *Operations Research*, 42:201–212, 1994.

[17] Holger H. Hoos and Thomas Stützle. Local search algorithms for sat: An empirical evaluation. *Journal of Automated Reasoning*, 24:421–481, 2000.

[18] Holger H. Hoos and Thomas Stützle. *SAT20000: Highlights of Satisfiability Research in the year 2000*, chapter SATLIB: An Online Resource for Research on SAT, pages 283–292. Frontiers in Artificial Intelligence and Applications. Kluwer Academic, 2000. (web site: http://www.satlib.org).

[19] Henry 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.

[20] Daniel Le Berre. Exploiting the real power of unit propagation lookahead. In *Proceedings of the Workshop on Theory and Applications of Satisfiability Testing (SAT2001)*, Boston University, Massachusetts, USA, June 14th-15th 2001. to appear.

[21] Chu-Min Li. A constrained based approach to narrow search trees for satisfiability. *Information processing letters*, 71:75–80, 1999.

[22] Chu-Min Li. Integrating equivalency reasoning into davis-putnam procedure. In *the proceedings of AAAI-2000*, pages 291–296, 2000.

[23] Chu-Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of IJCAI'97*, pages 366–371, 1997.

[24] G. Logeman M. Davis and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, pages 394–397, 1962.

[25] João P. Marques-Silva and Karem A. Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[26] David G. Mitchell and Hector J. Levesque. Some pitfalls for experimenters with random sat. *Artificial Intelligence*, 81(1-2):111–125, March 1996.

[27] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, June 2001.

[28] Allen van Gelder and Yumi Tsuji. Instances from circuit fault analysis. Available in [11] repository.

[29] Allen VanGelder and Fumiaki Kamiya. The partial rehabilitation of propositional resolution. Technical Report UCSC-CRL-96-04, 1996.

[30] Miroslav N. Velev. Formal verification of superscalar and VLIW processors benchmarks (FVP-UNSAT.1.0). Available at http://www.ece.cmu.edu/~mvelev.

[31] Available for downloading at http://sat.inesc.pt/benchmarks/cnf/uni-paderborn.

[32] Hantao Zhang. SATO: An efficient propositional prover. In *CADE-14*, LNCS 1249, pages 272–275, 1997.