SatEx: Towards an Exhaustive and Up-to-Date SAT Experimentation

Laurent Simon
Laboratoire de Recherche en Informatique,
U.M.R. CNRS 8623, Université Paris-Sud,
91405 Orsay Cedex, France
phone: +33 (0)1 69 15 64 95, fax: +33 (0)1 69 15 85 86,
e-mail: simon@lri.fr, web: www.lri.fr/˜simon

Abstract

The SatEx web site is devoted to experimentation around SAT. It is not only a front-end to a database gathering an exhaustive number of executions, but it also allows dynamic result synthesis as well as a detailed exploitation of all experimentations. Being dynamically generated and constantly evolving, this site can be considered as an always up-to-date SAT experimentation *paper*. At this time, the first realease of the SatEx (http://www.lri.fr/˜simon/satex/satex.php3) is already used by the SAT community, with more than 15000 hits in a few months. The new release will present the results of more than 450 cpu-days on a recent machine and will incorporate some major improvements.

1 Introduction

Experimentation takes an increasing place in AI and especially in the particular area of SAT, where it is widely used to compare algorithms and implementations. Nine years ago, the Dimacs competition, bringing together a number of different solvers and benchmarks, has provided a strong and lasting interest in SAT algorithms. This kind of competition, with an increasing number of real life problems encoded using the poor-but-efficient SAT formalism, has driven authors to constantly improve their algorithms. Since this fruitful meeting, the number of benchmarks and solvers is growing up [12] in such a way that, if the first Dimacs competition took only some days to complete, this wouldn't be the case today anymore. For instance, if one wants to gather all benchmarks and solvers in order to run each combination of them, then such a competition would take more than one year today, even on a very recent computer. Thus, achieving a satisfying experimental analysis is not easy and is very cpu-time consuming. One common solution is to compare only a few number of programs on a limited number of benchmarks. But, in this case, the experimentation process is biased by the human point of view on program performances.

The main ideas of our web site are the following: first, we want to propose a new experimentation framework, in which anybody could systematically check every parameters and results of each execution in order to easily recreate the same experimentation conditions as those presented in papers. Who has never failed reproducing results reported in a paper? Secondly, we want to share the cpu-time consumption of this work by allowing one to explore by himself our own base of results without running again the same couple of program/benchmark. We provide all experimentations in exactly the same format that one would have obtained by himself. Anybody can thus compare his program in a more satisfying and complete way with a minimum effort: just add it to the database.

2 A Quick SatEx Guided Tour

At the submission date, the SatEx release 1.0 is already online and fully operational. Based on our experience of this version, we propose to add some new functionalities. This new release (2.0) represents more than 450 cpu-days[1] of experimentations, in more than 30000 executions traces, in a 200 Mb database. The cut-off time given for an execution is 10000 s. The SAT provers used are:

- DLL [15] ones: `asat` and `csat` [9], `eqsatz` [13], `nsat` and `sat-grasp`, [16], `posit` [10], `relsat` [1], `sato` and `sato-3.2.1` [18], `satz` and `satz-213` [14];
- DP [6] ones: `calcres` [2], `dr` [7] and `zres` [4; 3];
- Randomized-DLL ones: `ntab` (with `ntab-back` and `ntab-back2`) [5], and `relsat-200` [1];
- Other approaches: `heerhugo` [11] and `modoc` [17].

The database currently contains 1488 benchmarks grouped in the following families: `aim`, `ais`, `barrel`, `beijing`, `bf`, `blockworld`, `dubois`, `f`, `fvp`, `g`, `hole`, `ii`, `jnh`, `joao`, `longmult`, `massacci`, `pader-easy`, `pader-hard`, `parity`, `pret`, `quasigroup`, `queueinvar`, `satplan`, `ssa`, `sw100-8-0`, `sw100-8-1`, `ucsc-bf` and `ucsc-ssa`
Among the new (and old) functionalities available in the SatEx 2.0 release, the most important are:

- Given a subset of programs and a subset of families of benchmarks, one can generate its automatic synthesis.

[1]On the Dimacs [8] 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%.

- View this synthesis with different format (Sum of cpu-time, Mean (with standard deviation), Median, 50% Interval of cpu time).
- View the exact command line parameter used and the output produced by the program for every launch.
- View a detailed summary of all executions of a program on a family of benchmarks.
- Sort algorithms depending on their performances on each families of benchmarks: sum, mean, median.
- Find automatically some kind of bugs (wrong or inconsistent program answer).
- View/Add comments on results (attached on a program and/or a family of benchmarks).
- Extract automatically up-to-date *challenging* benchmarks (*e.g.* benchmark not solved yet by any program).

From a technical point of view, the internal architecture of the SatEx consists of three databases. One focusing on the web-presentation of results, answering user queries. The second is an off-line copy of the first, used for checking new updates before releases. The last database is the working one, isolated from the web for security purposes, and used for job submission, benchmarks and program additions. The main upgrade of the SatEx regards this last database, in order to allow different machines to request jobs. For this, a special job server that dispatches jobs over a network of 20 biprocessor Linux workstation has been designed. This upgrade was needed in order to face the incredible cpu-ressource needed to maintain SatEx. It also allows randomized algorithm to be added, where thousands of runs are needed for only one benchmark. If the current release 2.0 only contains results from the same computer, this upgrade will also allow to maintain the database up-to-date with the constantly evolving computer performances: we can launch again all failed executions on more recent machines with the same cut-off parameter, and automatically upgrade the results of the whole database by scaling them up to some new reference machine. On this topic, we are currently working on how to scale-up different machines in a satisfying way.

3 Conclusion

For now, only complete algorithms are well considered in the SatEx. Taking randomization and uncomplete algorithm into account is one of the big improvements of the site, even if this had already been considered during the database architecture design. We also plan to add other functionalities, such as allowing job submission through the web, by submitting benchmarks and/or programs, and allowing an easy download of results for an easier (and local) consult.

We think that, for incrementality, homogeneity, transparency, programs/benchmarks distribution and programs behavior study (which is the basis of experimentation in other areas), a project like the SatEx is necessary. Moreover, we think that a project like our site should be used in other AI fields than SAT. Experimentation in AI could use some general framework, based on a double publication. One using “regular” paper, and the other one using the web. 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 fair and unbiased results to be published.

References

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

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

[3] Ph. Chatalic and L. Simon. Multi-resolution on compressed sets of clauses. In *12th ICTAI*, pages 2–10, 2000.

[4] Ph. Chatalic and L. Simon. Zres: the old dp meets zbdds. In *Proceedings of the 17th CADE*, pages 449–454, 2000.

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

[6] M. Davis and H. Putnam. A computing procedure for quantification theory. *JACM*, pages 201–215, 1960.

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

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

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

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

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

[12] Holger H. Hoos and Thomas Sttzle. *SAT'2000, SATLIB: An Online Resource for Research on SAT*, pages 283–292. IOS Press, 2000. (web site: http://www.satlib.org).

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

[14] C.M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of IJCAI'97*, pages 366–371, 1997.

[15] G. Logeman M. Davis and D. Loveland. A machine program for theorem-proving. *CACM*, pages 394–397, 1962.

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

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

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