Learning Back-Clauses in SAT
(Poster Presentation)

Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann

IBM Watson Research Center, Yorktown Heights, NY 10598, USA
{ashish.sabharwal,samulowitz,meinolf}@us.ibm.com

In [3], SAT conflict analysis graphs were used to learn additional clauses, which we refer to as *back-clauses*. These clauses may be viewed as enabling the powerful notion of "probing": Back-clauses make inferences that would normally have to be deduced by setting a variable deliberately the other way and observing that unit propagation leads to a conflict. We show that short-cutting this process can in fact improve the performance of modern SAT solvers in theory and in practice. Based on out numerical results, it is suprising that back-clauses, proposed over a decade ago, are not yet part of standard clause-learning SAT solvers.

Back-Clauses. We assume familiarity with SAT conflict analysis [3, 4]. Figure 1 shows an example formula and its conflict graph derived after branching on (-1), (+2), and (+3). A clause such as (1-2+5) in our notation may be thought of as $(x_1 \vee \neg x_2 \vee x_5)$. The corresponding first or rightmost UIP at the decision level is literal (-10) and the standard clause learnt from this conflict is (-4+10). In [3] it was found that, for any two consecutive UIPs at level L, we can infer that, under some *context* given by the literals on tree levels $< L$, the left UIP implies the right UIP. In our example, given 5, 7 implies -10. Written as a clause, this gives (-5-7-10). It makes sense to add this "back-clause" because unit propagation is incomplete and may in fact not be able to infer that, given 5, 10 implies -7. In our example, we can also infer that, given 2, 3 implies 7, or (-2-3+7). Note that these two clauses imply (-2-5-3-10), also under incomplete unit propagation. Since back-clauses in general have smaller "contexts" than traditional nogoods based on *all* UIPs, we conclude from the following proposition that adding all back-clauses between adjacent UIPs at level L is, in general, strictly stronger under unit propagation than adding all UIP nogoods at level L.

A. (1+4)	B. (1-2+5)
C. (-2-3-6)	D. (-3+6+7)
E. (-4+10+11)	F. (-5-7+8)
G. (-5-7+9)	H. (-8-9-10)
I. (10-11)	

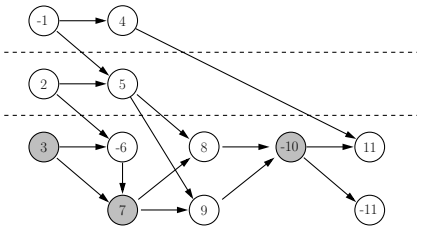

Fig. 1. An Example Formula and its Conflict Graph

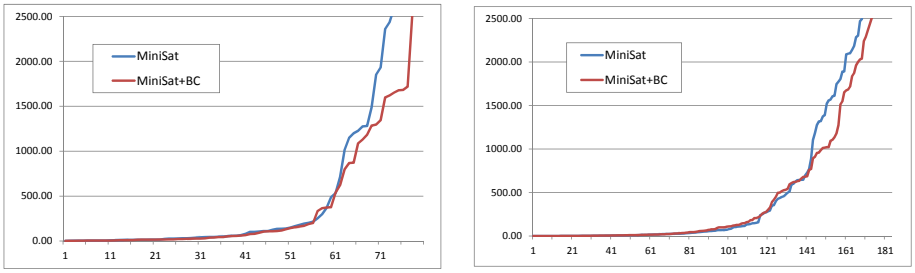

Fig. 2. Cactus plot showing the maximum time (y-axis, in seconds) needed by `MiniSat` with and without Back-Clauses to solve a given number of instances (x-axis). Left: SAT Race 2010 benchmark. Right: SAT Competition 2011 benchmark.

Proposition 1. *By adding the first UIP clause and back-clauses between every two consecutive UIPs at level L, we enable unit propagation to make all inferences that* all *traditional nogoods based on* all *UIPs at level L would.*

Empirical Evaluation. We added back-clause lerning as part of version 2.2.0 of `MiniSat` [5] and experimented with it on 2.3 GHz AMD Opteron 6134 machines with eight 4-core CPUs and 64 GB memory, running Scientific Linux release 6.1. As benchmarks we use all of the application instances from the 2010 SAT Race and the 2011 SAT Competition. Both `MiniSat` and `MiniSat+BC` (i.e., with back-clauses on the decision level) were configured to first simplify the formula using the `SatELite` preprocessor [2]. Figure 2 summarizes the results in terms of the commonly used cactus plot metric. We observe that learning back-clauses is clearly helpful for both benchmark sets, particularly for harder instances where the benefits of learning additional clauses become most noticeable.

In conclusion, we rediscovered the idea of learning back-clauses during search, first introduced in [3]. We showed that adding back-clauses is stronger than adding no-goods for all UIPs. We hope that our experiemntal findings will help this technique find its rightful place among modern SAT solving methods.

References

1. R.J.J. Bayardo and R.C. Schrag Using CSP look-back techniques to solve real-world SAT instances. *In Proceedings of AAAI'97, 203–208, 1997.*
2. N. Een and A. Biere. Effective preprocessing in sat through variable and clause elimination, *In proc. SAT'05, volume 3569 of LNCS, 61–75, 2005.*
3. J.P. Marques-Silva and K.A. Sakallah. GRASP: A Search Algorithm for Propositional Satisfiability, *IEEE Trans. on Computers 48(5):506-521.*
4. M.W. Moskewicz and C.F. Madigan and Y. Zhao and L. Zhang and S. Malik. Chaff: Engineering an Efficient SAT Solver, *In Proc. DAC, 530–535, 2001.*
5. N. Sorensson and N. Een. MiniSAT 2.2.0, *http://minisat.se*, 2010.
6. SAT Competition. *http://www.satcomptition.org.*