Ben-Jose SAT solving software library.

Jose Luis Quiroga. 2014.

https://github.com/joseluisquiroga/ben-jose

JoseLuisQuirogaBeltran@gmail.com

Abstract. The software library Ben-Jose for solving instances of the satisfiability problem in CNF form is presented. Ben-Jose implements a trainable strategy that extends the traditional DPLL+BCP+CDCL resolution based approach, first introduced by Joao Marquez da Silva [01] and latter refined by others [14][15], with an original technique to check if the structure of a sub-formula of the solving SAT instance has previously been found to be unsatisfiable, in order to skip the search on it whenever found again. The technique introduces an original stabilization procedure (as in [16]) for the structure of the sub-formulas that is tightly coupled with the work done by BPC (each BCP step groups some variables) and has linear complexity.

Introduction.

The SAT problem is the canonical decision problem by excellence [02][03][04]. It lays at the heart of the P vs NP question and its importance cannot be overstated [05]. People have proved both that “NP != P” and that “NP==P” [06] [07], and filed patent applications for optimal SAT solvers based on resolution [08].

Exponential lower bounds of resolution (RES) proof size for the pigeon hole principle (PHP) have been proven [09] [10], and polynomial size proofs for extended resolution (ER) have also been proven for the PHP [11] [12] [13] instances of the SAT problem. That work suggests that solvers based on RES [01] [14] [15] need “something else” in order to be “faster” [17][18].

Based on the notion that theorems are proved with lemmas and the observation that the structure of PHP(n+1, n) can be matched with several substructures of PHP(n+2, n+1), the software library presented in this work (Ben-Jose) extends RES by learning the structure of unsatisfiable sub-formulas (proved unsatisfiable during its execution) and matching them against future structures of sub-formulas found during its execution, in order to skip the search, and directly backtrack on them, when ever a match (subsumed isomorphism) is found.

This technique is here after called Backtrack Driven by Unsatisfiable Substructure Training (BDUST). BDUST introduces an original stabilization procedure (as in [16]) for the unsatisfiable sub-formulas found, uses the work done by BPC (each BCP step groups some variables), and has linear complexity with respect to the size of the sub-formulas. The structures learned with BDUST have the advantage that can be used also with different instances than the one they were found on. That is why BDUST is “training” and not “learning”.

This form of extending RES is not ER, as the system presented by Tseitin [19], because the subsumed isomorphism detection technique is not RES based, and the power and complexity of the resulting proof system has not been formally studied. The empirical results on the classical problems of PHP and isPrime (based on the Braun multiplier) are presented here.

The Library

The following classes and names for attributes are the most important to explain how the solver works. They are explained in terms used in [01], [14] and [16].

DPLL+BCP+CDCL classes.

To explain the most important parts of DPLL+BCP+CDCL, the following classes will be used.

neuron. class for CNF clause behavior. So there is one neuron per clause.

quanton: class for CNF variables (each variable has a positon and a negaton). There are two quantons per variable. Neurons hold references to quantons called fibres. They are used for BCP.

prop_signal: class for representing BCP propagation data: which quanton fired by which neuron (which clause forced a given variable). BCP in done with the two watched literals technique (two watched fibres in the library's terminology).

deduction: class for holds the result of analyzing (doing resolution) of a conflict. It has the data for learning new neurons (clauses).

brain: class that holds all data used to solve a particular CNF instance. So there is one brain per CNF instance.

deducer: class that holds the data used to analyze a conflict.

leveldat: A level is all that happens between choices during BCP. So there is one level per choice. This class holds level relevant data.

Stabilization classes

The following classes will be used to explain the most important aspects of CNF stabilization:

sort_glb: holds all global data used to stabilize (kind of sorting) a group of items (neurons and quantons representing a sub-formula of a CNF).

sortee: an item to be stabilized. Neurons and quantons contain sortees that are used to stabilize CNF sub-formulas.

sorset: in order to stabilize a group of items the sort_glb class (or sortor) needs to group items (neurons and quantons in our case) in several iterations. This class is used for such iterated sub grouping.

sortrel: class used to establish the relations between the sortees to be stabilized. They must be properly initiated before each stabilization. They define the sub-formula in partucular to be stabilized by defining the relations between a particular sub set (sub-formula) of neuron sortees and quanton sortees.

Matching classses

The following classes will be used to explain the most important aspects of CNF matching:

coloring: The initial and final state for an stabilization is a coloring: a grouping of sortees where each colors identifies a group of sortees. A complete coloring is one in which there is one color for each sortee, so that each group has exactly one item (one sortee). This class is used to specify only the input to the stabilization process (the initial state). The class canon_cnf is used for the output (it is the result of applying the output coloring to the stabilized sub-formula ordered by color). To initialize the sortor, it “loads” the initial coloring into the sort_glb instance that will stabilize the CNF sub-formula.

canon_cnf: This class is used to represent the output of an stabilization process: the stabilized CNF sub-formula. It is the interface class to the database class that handles all disk operations (the skeleton class). This class contains some disk handling related information (paths and sha info). A canon_cnf basically is a set of canon_clauses (which are basically arrays of numbers).

memap: This class represents a CNF sub-formula. It is the pivot class to do all stabilization. It is matained during BCP and used during backtracking in order to know what CNF sub-formulas are to be stabilized and searched for in the database (skeleton class). There is one memap per leveldat and they are either active or inactive. Active when they are condidates for stabilization, matching and search in database (or saving), at backtrak time. When a CNF sub-formaula, during search, is found to be unsatisfiable, is not trivial (BCP could not figure it out), and both search branches had the same variables (so that it can latter be searched only with one of them), it is saved, stored in the database (skeleton class). Every time an still active memap has done its first branch of BCP, it is stabilized and searched for in the database (skeleton class). Trivial sub-formulas are called anchors in the code because they serve as a start point for stabilizing not trivial ones.

Database classes

The skeleton class handles all disk related functions and management. The database is basically a directory in disk. In that directory unsatisfiable canon_cnf are saved and searched by the SHA function of their content. They are saved in a path that is constructed with the SHA and other relevant search info.

Since an unsatisfiable sub-formaula might not be minimal (have some unnecessary clauses for unsatisfiability), each unsatisfiable CNF sub-formula has three relevant canon_cnf:

The guide. It is the canon_cnf resulting of stabilizing the CNF sub-formula covered by first search branch variables. So it is a satisfiable part of the unsatisfiable CNF sub-formula that is a “guide” for the search.

The tauto. It is the full unsatisfiable CNF sub-formula. It is the canon_cnf resulting of stabilizing the CNF sub-formula covered by both search branches charged quantons (used variables).

The diff. This canon_cnf contains all canon_clauses in tauto but not in guide. Each diff is saved in a path called 'variant' in the skeleton. So one guide can have several variants.

A search of a target CNF sub-formula is conducted in two phases: the search for the guide of the target and the search for the variant that is a sub-formula of the target diff. Once the guide is stabilized the search for it is a simple: “see if its path exists” (remember that its path contains the SHA of its content). If the target canon_cnf is not equal to a variant (the path does not exist), the second phase is more time consuming because it involves reading each variant and comparing it to the target diff to see if the the variant is a sub-formula of the target diff (which would mean that the target is unsatisfiable and therefore can be backtracked).

References

[01] Joao Marques da Silva. Search Algorithms for Satisfiability problems in combinatorial switching circuits, 1995. A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Electrical Engineering). The University of Michigan.

[02] Armin Biere. Handbook of Satisfiability, 2009.

[03] Daniel Kroening, Ofer Strichman. Decision Procedures: An Algorithmic Point of View, 2008.

[04] Victor W. Marek. Introduction to Mathematics of Satisfiability, 2009.

[05] Stephen Cook. The importance of the P versus NP Question, 2003. JACM 50, 1 (50th Anniversary Issue), pp 27-29.

[06] GJ Woeginger. The P-versus-NP page, 2014. http://www.win.tue.nl/~gwoegi/P-versus-NP.htm. .

[07] Matthias Muller. Polynomial SAT-Solver, 2013. http://vixra.org/pdf/1212.0109v2.pdf. (this one completely fools me).

[08] J. L. Quiroga. Optimal circuit verification method, 2001. http://www.google.com/patents/US20040250223. (shame on him).

[09] A Haken. The intractability of resolution, 1985. Theoretical computer science, 39, pp. 297-308..

[10] Samuel R. Buss, Gyorgy Turan. Resolution Proofs of Generalized Pigeonhole Principles, 1988. .

[11] S. A. Cook. A short proof of the pigeon hole principle using extended resolution, 1976. SIGACT News 8(4):28-32..

[12] S. A. Cook, R. A. Reckhow. The relative efficiency of propositional proof systems, 1979. Journal of Symbolic logic. 44, pp. 25-38.

[13] Matti Jarvisalo. Impact of Restricted Branching on Clause Learning SAT solving, 2007. Research Reports 107. pp.28. Helsinki University of Technology Laboratory for Theoretical Computer Science.

[14] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, Sharad Malik. Chaff: Engineering an Efficient SAT Solver, 2001. DAC 01. Proceedings of the 38th annual Design Automation Conference.

[15] Niklas Een, Niklas Sorensson. An Extensible SAT-solver, 2004. Chalmers University of Technology, Sweden.

[16] Oliver Bastert. Stabilization Procedures and Applications, 2002. Zentrum Mathematik. Technische Universitat Muchen.

[17] Heidi E. Dixon, Matthew L. Ginsberg, David Hofer, Eugene M. Luks, Andrew J. Parkes. Generalizing Boolean Satisfiability I, II and III: Background and Survey of Existing Work, Theory and Implementation, 2004. Journal of Artificial Intelligence Research 21, 22 and 23.

[18] Gilles Audemard, George Katsirelos, Laurent Simon. A Restriction of Extended Resolution for Clause Learning SAT Solvers, 2010. www.aaai.org.

[19] G. Tseitin. On the complexity of proofs in propositional logics, 1983. Automation of Reasoning: Classical Papers in Computational Logic 1967-1970. volume 2. Springer- Verlag.