

kcnfs

Gilles Dequen
 LaRIA CNRS FRE-2733,
 Univ. de Picardie Jules Verne
 33, rue St Leu
 80039 Amiens Cedex 1
 email: gilles.dequen@u-picardie.fr

Olivier Dubois
 LIP6 CNRS
 Univ. Paris 6,
 4, Place Jussieu
 75252 Paris cedex 5
 email: Olivier.Dubois@lip6.fr

Abstract

kcnfs is a DPLL based solver. *kcnfs* integrates a branching variable heuristic devoted essentially to prove the unsatisfiability of random k -SAT formulae and which has been inspired by recent statistical physics work. This heuristic was presented in [DD01] and generalized to solve k -SAT formulae with a renormalization strategy in [DD03]. The local techniques called "picking" first described in [DD01] were implemented in *kcnfs*.

kcnfs

In [DD01], we presented a new branching variable selection heuristic for solving random 3-SAT formulae using a DPLL-type procedure. We implemented our heuristic in a solver named *cnfs*. We generalized this heuristic for solving random k -SAT formulae ($k \geq 3$) in a new implementation named *kcnfs* [DD03]. The general ideas having led to the development of *cnfs* and *kcnfs* are the following.

The intuitive idea behind *BSH* is that a variable of the backbone of a formula (if it exists) belongs to some cycles of a hypergraph for which the vertices correspond to the literals and the hyperedges correspond to the clauses of the formula. A variable which belongs to two symmetric cycles can be set neither to *true* nor to *false* without producing a contradiction. In the following subset of clauses, from a given formula: $(a \vee b \vee c) \wedge (\bar{c} \vee d \vee e) \wedge (\bar{e} \vee a \vee \bar{e}) \wedge (\bar{b} \vee f \vee g) \wedge (\bar{g} \vee \bar{d} \vee e) \wedge (a \vee \bar{d} \vee g) \wedge (c \vee d \vee \bar{f}) \wedge (a \vee \bar{b} \vee \bar{e}) \wedge (d \vee \bar{f} \vee \bar{g}) \wedge (\bar{b} \vee c \vee e)$, the literal \bar{a} belongs to a cycle of the associated hypergraph, and then if it is set to *true* a contradiction occurs. It is a backbone variable for this set of

clauses. The backbone variables play a crucial role for the size of a refutation tree. If the literal a , in the previous sample, is chosen as the root of a subtree, as in the one on the left in Figure 1, the contradiction generated by setting a to false is detected only once. On the contrary, if a is chosen as an intermediate node, as in the subtree on the right in Figure 1, then the contradiction due to $a = false$ is detected several times.

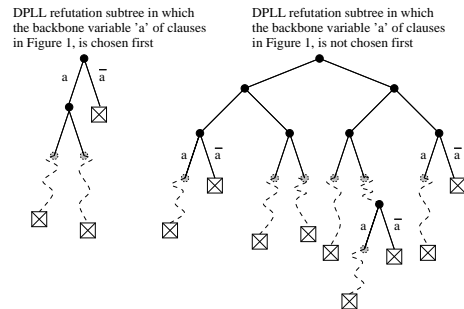


Figure 1: Typical DPLL refutation tree according to whether the literal a of the set of clauses previously described is given top priority or not for branching

Let t be a variable of \mathcal{F} . t and \bar{t} are the positive and negative literals respectively, associated to the variable t . The main idea behind the *BSH* heuristic is as follows. *BSH*(t) aims to measure the correlations of the literal t with all the other variables of the formula through the clauses where t appears. In a practical way, the measure of the correlation of a literal t with other variables is viewed as an estimation of the number of possibilities that the literal t is forced to *true*, lest a contradiction occur, as a function of the truth values assigned to the other variables.

The measure of correlation estimation of a lit-

eral t belonging to a random 3-SAT formula is computed as follow:

$BSH(t)$

- (i) Built the set of binary clauses, any of which being *false* forces t to be *true* unless a contradiction occurs, directly or through the unit propagation process., This set is noted $\mathcal{I}(t)$
- (ii) For each clause $C_i : u_i \vee v_i$ from $\mathcal{I}(t)$ compute the product $P_i : \mathcal{M}(u_i) \times \mathcal{M}(v_i)$ where $\mathcal{M}(t)$ can be:
 - $((2p_2(t) + p_3(t))$ where $p_2(t)$ (resp. $p_3(t)$) is the number of occurrences of t in binary (resp. ternary) clauses.
 - $BSH(t)$ if one evaluates with a more accurate estimation.
- (iii) Sum all P_i

Let us take an example showing how to compute $BSH(t)$ with a single level of evaluation for the following set of clauses: $(t \vee \bar{a} \vee b) \wedge (t \vee c) \wedge (\bar{c} \vee d \vee \bar{e}) \wedge (a \vee f \vee \bar{d}) \wedge (a \vee g) \wedge (\bar{b} \vee f \vee g) \wedge (\bar{d} \vee e)$. We have $\mathcal{I}(t) = \{\bar{a} \vee b, d \vee \bar{e}\}$. The clause $\bar{a} \vee b$ from $\mathcal{I}(t)$ is obtained from the first clause of our sample. The clause $d \vee \bar{e}$ belongs to $\mathcal{I}(t)$ from the third clause of our sample and through unit propagation of the second one. Thus, for the variables in the clauses of $\mathcal{I}(t)$, the value of $BSH(t)$ is : $(2p_1(a) + p_2(a))(2p_1(\bar{b}) + p_2(\bar{b})) + (2p_1(d) + p_2(d))(2p_1(e) + p_2(e)) = (2+1)(1) + (2+1)(2) = 9$.

The chosen branching variable t , is the one which has the highest score $BSH(t) \times BSH(\bar{t})$. There are two limitations which must restrict the use of the heuristic BSH for an efficient implementation. First it can be noticed that if at a current node of the solving tree the sub-formula associated with the considered node has many binary clauses, the computation of $BSH(t)$ can be too time-consuming with respect to simpler heuristics essentially based on the binary clauses. In *kcnfs* a limitation has been defined empirically as a function of the proportion of binary clauses in a sub-formula. The second limitation concerns the level of evaluation of $BSH(t)$. A good limitation must be a compromise between the accuracy of the evaluation which must remain significant so that it is not distorted by variables taken into account more than one time and the cost of the computation. In *kcnfs* this limitation has been defined empirically as a function of the number of variables and the maximum length of the clauses in

the formula.

BSH can be generalized to solving k -SAT formulae and more generally, to process formulae with clauses having various lengths. to preserve a sound estimation, we have to normalize the BSH evaluation of a z_1 -clause compared to a z_2 -clause with $z_2 > z_1$. The renormalization process consists in adding a virtual literal as many times as necessary so that all clauses in $\mathcal{I}(t)$ have the same length. Let us take an example showing specifically how the renormalization is done. Consider that t appears in the two following clauses : $\{(t \vee \bar{a} \vee b \vee d), (t \vee c \vee e)\}$, then the set $\mathcal{I}(t) = (\bar{a} \vee b \vee d), (c \vee e)$. The renormalization consists in adding a virtual literal as many times as necessary so that all clauses in $\mathcal{I}(t)$ have the same length. In the present example, α is added to $(c \vee e)$ giving $(c \vee e \vee \alpha)$. Then the value associated with the virtual literal α , is the mean of the evaluation of literals of $\mathcal{I}(t)$ i.e. a, b, c, d, e .

Some local treatments, named “picking” [DD01], are implemented in *kcnfs* to help $BSH/RBSH$ in selected for branching “backbone variables”. Picking is available when the unit propagation is processed. When k increases, the unit propagation process rate drastically decreases because of the difficulty to produce binary clauses. Moreover, if it exist some binary clauses in the formula, the unit propagation process is often not good enough to have a successful deduction with picking. Within a practical framework, picking cannot help $RBSH$ when $k > 3$. However, *kcnfs* obtains good performances on solving k -SAT formulae with its only branching rule.

References

- [DD01] O. Dubois and G. Dequen. A Backbone Search Heuristic for Efficient Solving of Hard 3-SAT Formulae. In *Proc. of the 17th Internat. Joint Conf. on Artificial Intelligence*, pages 248–253, Seattle, August 4–10 2001.
- [DD03] G. Dequen and O. Dubois. *kcnfs*: An efficient solver for random k -SAT formulae. In *International Conference on Theory and Applications of Satisfiability Testing (SAT), Selected Revised Papers, LNCS*, volume 6, pages pp 486–501, may 2003.