J Autom Reasoning (2006) 37:261–276
DOI 10.1007/s10817-006-9025-2

An Efficient Approach to Solving Random *k*-SAT Problems

Gilles Dequen · Olivier Dubois

Published online: 20 October 2006

Abstract Proving that a propositional formula is contradictory or unsatisfiable is a fundamental task in automated reasoning. This task is coNP-complete. Efficient algorithms are therefore needed when formulae are hard to solve. Random *k*SAT formulae provide a test-bed for algorithms because experiments that have become widely popular show clearly that these formulae are consistently difficult for any known algorithm. Moreover, the experiments show a critical value of the ratio of the number of clauses to the number of variables around which the formulae are the hardest on average. This critical value also corresponds to a 'phase transition' from solvability to unsolvability. The question of whether the formulae located around or above this critical value can efficiently be proved unsatisfiable on average (or even for a.e. formula) remains up to now one of the most challenging questions bearing on the design of new and more efficient algorithms. New insights into this question could indirectly benefit the solving of formulae coming from real-world problems, through a better understanding of some of the causes of problem hardness. In this paper we present a solving heuristic that we have developed, devoted essentially to proving the unsatisfiability of random *k*SAT formulae and inspired by recent work in statistical physics. Results of experiments with this heuristic and its evaluation in two recent SAT competitions have shown a substantial jump in the efficiency of solving hard, unsatisfiable random *k*SAT formulae.

Key words satisfiability · solving · heuristic

G. Dequen (✉)
LaRIA, Université de Picardie Jules Verne, 33 Rue St Leu, 80039 Amiens Cedex 1, France
e-mail: gilles.dequen@u-picardie.fr

O. Dubois
LIP6, CNRS-Université Paris 6, 4 place Jussieu, 75252 Paris Cedex 05, France
e-mail: Olivier.Dubois@lip6.fr

1. Introduction

The past decade has seen a fast growth of theoretical as well as algorithmic investigations on random *k*SAT formulae [3–5, 11, 17, 19, 20, 26, 34]. A random *k*SAT formula consists of a subset of clauses chosen uniformly, independently and with replacement, from the set of all possible clauses containing k distinct literals over a set of n variables. Indeed, these formulae provide a rich model for exploring many aspects of computing complexity. Even a link has been recently established between average-case complexity and approximation complexity using random 3SAT formulae [14]. Random formulae have, in contrast with formulae from the real world, some typical structure that can be characterized by parameters giving means to determine the source of the difficulty of solving these formulae. For example, it is now well known that clause density, that is, the ratio of the number of clauses to the number of variables, concentrates the hardest formulae on average around a critical value where the probability that a formula has a solution drops rapidly from 1 to 0, producing a transition from solvability to unsolvability. Also, it appears that it is easier for random formulae to capture and to analyze features relevant to computational complexity [2, 5, 13, 30, 35]. On the other hand, in the processing of formulae from the real world, a part of the difficulty of solving can come from features specific to the original problem [1, 12, 18]. The structural advantages that random SAT formulae appear to have probably account for the vast number of studies that they elicit. Confirming this trend, over the past few years statistical physicists have shed new light on these random formulae by focusing on the structure of the space of solutions. Let us mention that a 'solution' from the physicists' point of view can be, in a general sense, an assignment to the variables of the formula that falsifies the minimum number of clauses. They showed that variables could become locally or globally 'frozen' (i.e., have a unique value in some groups or all solutions) as a function of parameters, foremost among which is the ratio of clauses to variables. These frozen variables lead to a well-defined structure of the space of solutions in a typical random formula. Such studies have led to theoretical as well as empirically convincing results [4, 27, 28]. The new vision brought by these statistical physics studies has inspired us to design a new heuristic for a DPLL-type algorithm taking into account the structure of the space of solutions.

It has been observed that in recent years, significant progress, sometimes dramatic, has been achieved for satisfiable random formulae below the satisfiability threshold [4, 19, 32]. On the other side, for random unsatisfiable formulae around or above the threshold, advances observed in the same period have seemed much more difficult [6, 9, 16, 24, 25]. Some misgivings had even been expressed as to the actual feasibility of any significant improvement, for example, solving hard random unsatisfiable formulae up to 700 variables. Yet, efficiently proving unsatisfiability of formulae is of crucial importance in some fields such as automated reasoning, multiagent systems. In this respect, unsatisfiable hard random formulae are challenging and represent a valuable test-bed for new algorithms. As mentioned above, based on work in statistical physics, we developed a new heuristic for proving unsatisfiability, which has yielded a very significant increase in the efficiency of solving random unsatisfiable formulae, making it possible, for example, to handle formulae with 700 variables [31]. In the following sections of this paper, we first present an overview of how our new heuristic on 3SAT formulae works. We then show how it has been generalized for *k*SAT formulae and, more generally, for processing formulae with clauses having

various lengths. Finally, we present a summary of the performance of our heuristic as implemented in the solver named *kcnfs*, which we have developed. We provide an experimental comparison with a selection of the best current or previous solvers.

2. The Backbone-Search Heuristic

In [11], we presented a new branching variable selection heuristic for solving random 3SAT formulae using a DPLL-type procedure [7]. We implemented our heuristic in a solver named *cnfs*. We generalized this heuristic for solving random kSAT formulae ($k \geq 3$) in a new implementation named *kcnfs* [8]. The general ideas having led to the development of *cnfs* and *kcnfs* are the following.

Let $\mathcal{F}$ be a 3SAT formula, and consider a DPLL-type algorithm [7] for solving $\mathcal{F}$. Such an algorithm develops a solving tree until all clauses of $\mathcal{F}$ are satisfied by values *true* or *false* assigned to the variables of $\mathcal{F}$ at nodes of the search tree, or until no satisfying truth assignment has been found. It is called a complete algorithm because it is able to give an answer for unsatisfiable as well as satisfiable formulae. However, the approach to efficiently solve an unsatisfiable formula is quite different from finding a satisfying assignment of truth values to the variables. In order to solve unsatisfiable formulae efficiently, an implicit solving tree with as small a size as possible is to be constructed. This is exactly the way we designed the heuristic *BSH* that we describe below. 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 that belongs to two symmetric cycles can be set neither to *true* nor to *false* without producing a contradiction. In the subset of clauses, Figure 1, from a given formula, 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, Figure 1, is chosen as the root of a subtree, as in the one on the left in Figure 2, 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 2, then the contradiction due to $a = \mathit{false}$ is detected several times. Now we give a description of the heuristic *BSH*.

Figure 1 In the above set of 10 clauses the literal a has the value *true* in all satisfying assignments; a is called a backbone variable of these clauses.

$$
\begin{aligned}
C_1 &: (a \vee b \vee c)\\
C_2 &: (\bar{c} \vee d \vee e)\\
C_3 &: (\bar{c} \vee a \vee \bar{e})\\
C_4 &: (\bar{b} \vee f \vee g)\\
C_5 &: (\bar{g} \vee \bar{d} \vee e)\\
C_6 &: (a \vee \bar{d} \vee g)\\
C_7 &: (c \vee d \vee \bar{f})\\
C_8 &: (a \vee \bar{b} \vee \bar{e})\\
C_9 &: (d \vee \bar{f} \vee \bar{g})\\
C_{10} &: (\bar{b} \vee c \vee e)
\end{aligned}
$$

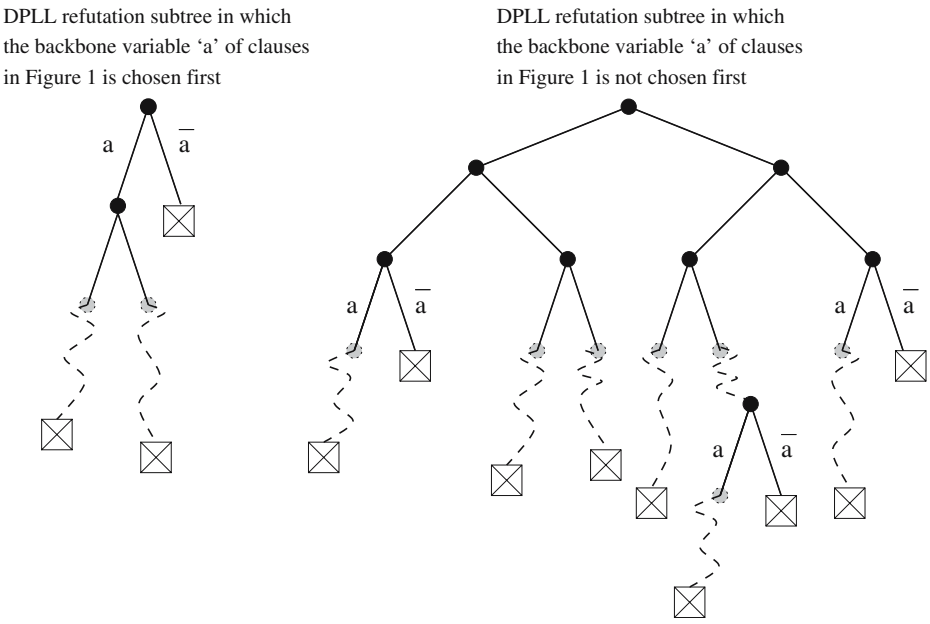

Figure 2 Typical DPLL refutation tree according to whether the literal a of the set of clauses in Figure 1 is given top priority or not for branching.

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 occurs, as a function of the truth values assigned to the other variables. More precisely, let us take, for example, a clause where the literal t appears : $(t \vee u \vee v)$. If both literals u and v can be *true* in most truth assignments satisfying many (possibly all) clauses, then t is a little correlated with the other variables. If one of the literals u and v must be *false* and the other one is *true* in most truth assignments satisfying many clauses, then t is a little more correlated with the other variables than in the previous case. Finally, if both literals u and v are *false* in most truth assignments satisfying many clauses, then t is strongly correlated to the other variables. Our heuristic is intended to identify the variables for which both associated literals are strongly correlated with the other variables, by means of the function $BSH(t)$ given in Figure 3, and described with a cinematic approach in the diagram, Figure 4. The branching variable chosen to be assigned successively *true* and *false* at a current node of the solving tree is one of the variables t having the highest score $BSH(t) \times BSH(\bar{t})$. The computation carried out by $BSH(t)$ is as follows.

The variable t is any literal not assigned a value at a current node of the solving tree and $BSH(t)$, its score. The evaluation of $BSH(t)$ is based on the set $\mathcal{I}(t)$ of binary clauses, any of which being *false* forces t to be *true* unless a contradiction occurs. There are two cases to be considered. The first case is that one of the occurrences

Figure 3 Function $BSH(t)$ of the 3SAT branching variable selection heuristic.

Set $i \leftarrow 0$, let t be a literal, and let MAX be an integer.

Integer $h(t)$

begin

$i \leftarrow i + 1$

compute $\mathcal{I}(t)$

IF $i < MAX$

return $\sum_{(u \vee v) \in \mathcal{I}(t)} h(\bar{u}) \times h(\bar{v})$ (1)

ELSE

return $\sum_{(u \vee v) \in \mathcal{I}(t)} (2p_2(\bar{u}) + p_3(\bar{u})) \times (2p_2(\bar{v}) + p_3(\bar{v}))$ (2)

END IF

end

of t appears in a ternary clause such as $(t \vee u \vee v)$. Then, if the binary clause $(u \vee v)$ is *false*, t is necessary *true*, so that $(t \vee u \vee v)$ is satisfied. The binary clause $(u \vee v)$ is therefore put in the set $\mathcal{I}(t)$. The second case is that one of the occurrences of t appears in a binary clause such as $(t \vee w)$. Then, one searches recursively for the binary clauses that can force $\bar{w}$ to be *true*. Such binary clauses consequently force t to be *true* through the unary clause w or through a chain of unary clauses. These binary clauses are also put in the set $\mathcal{I}(t)$. In this way, $\mathcal{I}(t)$ is the union of all binary clauses that, if *false*, will force t to *true* either directly or indirectly (through unary clauses). A sector of the circles in Figure 4 represents one binary clause of $\mathcal{I}(t)$ and takes part in the evaluation of $BSH(t)$ represented in the centers of the circles. The evaluation of $BSH(t)$ (independently of the level of accuracy) is the sum of the values computed in all sectors.

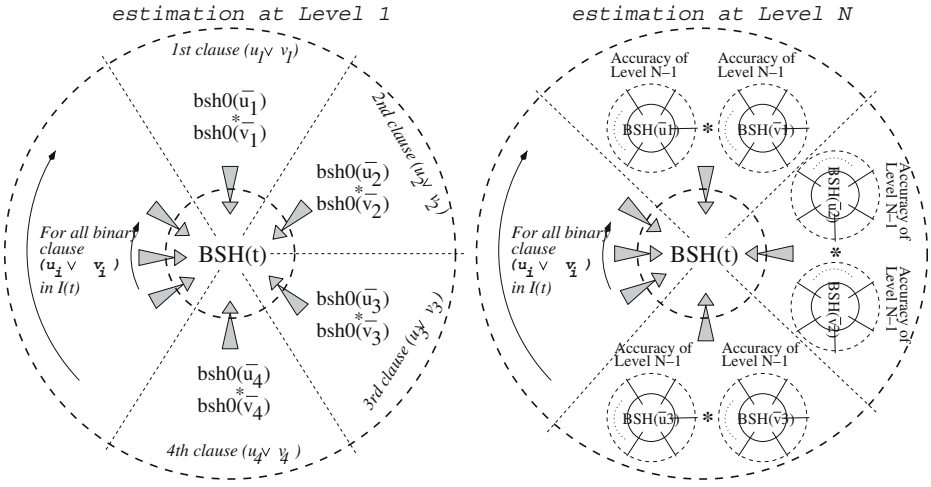

Figure 4 Cinematic description of the function $BSH(t)$ of the 3SAT branching variable selection heuristic.

$BSH(t)$ is evaluated at different levels of accuracy. The measure resulting from BSH is considered the more accurate as the level of recursive computation increases (see the circle on the right-hand side of Figure 4 and the line labeled (1) in Figure 3). Practically, this recursion is limited to an empirical value in order to avoid conflicting assignments at leaves of the recursive tree of computation. Nevertheless, if this case occurs before the limit value is reached, a control subroutine aborts the current BSH evaluation, and then the next candidate variable is evaluated. Moreover, BSH evaluation is applied only within the top part of the refutation tree, since it is time-consuming. This limitation is defined as a function of the proportion of clauses of length 2 among the remaining clauses. At level 1, $BSH(t)$ is given directly by the sum of the products computed in the sectors of the circle on the left-hand side of Figure 4, corresponding to the line labeled (2) in Figure 3. For a given literal x, let $bsh0(x)$ be the function $2 \times p_2(x) + p_3(x)$, where $p_2(x)$ and $p_3(x)$ are the numbers of binary and ternary clauses, respectively, and where x appears in the subformula at the current node. The weighting of p_2 by 2 is due to the fact that as a rough estimate (for a uniform distribution of assignments), x has twice the probability of being forced to *true* in a binary clause as against a ternary clause. Each product of sectors of Figure 4 (for the left-hand side and right-hand side estimations) represents the weighted number of possibilities that a binary clause $(u_i \vee v_i)$ of $\mathcal{I}(t)$ is *false* if one assumes that every occurrence of $\bar{u}_i$ and every occurrence of $\bar{v}_i$ are forced to be *true* in the clauses where $\bar{u}_i$ and $\bar{v}_i$ appear. To illustrate the computation of BSH at level 1, let us consider, for example, the subset of clauses from a 3SAT formula in Figure 5. The estimation $BSH(a)$ on this subset of clauses is described in Figure 6. For an estimation with level1 accuracy (see the top of the Figure 6), the set $\mathcal{I}(a)$ contains the clause $(d \vee e)$ derived from C_1 by a direct implication of a set to *false*, and the clause $(h \vee i)$ derived by successive implications from C_2, C_3, and C_4. The estimation of $BSH(a)$ with level2 accuracy is described at the bottom of Figure 6. On the whole, $BSH(t)$ may be interpreted as a measure of some correlation between t and all other variables that can force t to be *true*, in the subformula at the current node where $BSH(t)$ is being evaluated. This measure can be computed recursively at different levels of accuracy, as mentioned in the right-hand side circle of Figure 4. The estimation of $BSH(t)$ with level-N accuracy remains based on the set $\mathcal{I}(t)$ and

Figure 5 Set of clauses that BSH is applied to in Figure 6.

$$
\begin{aligned}
&\ldots\\
C_1 &: (a \vee d \vee e)\\
C_2 &: (a \vee f)\\
C_3 &: (\overline{f} \vee g)\\
C_4 &: (\overline{g} \vee h \vee i)\\
C_5 &: (\overline{h} \vee j)\\
C_6 &: (\overline{j} \vee k \vee l)\\
C_7 &: (\overline{i} \vee l \vee j)\\
C_8 &: (\overline{d} \vee \overline{b} \vee \overline{c})\\
C_9 &: (\overline{e} \vee \overline{h} \vee \overline{j})\\
C_{10} &: (\overline{j} \vee m \vee n)\\
C_{11} &: (\overline{j} \vee o \vee p)\\
C_{12} &: (\overline{e} \vee r \vee s)\\
&\ldots
\end{aligned}
$$

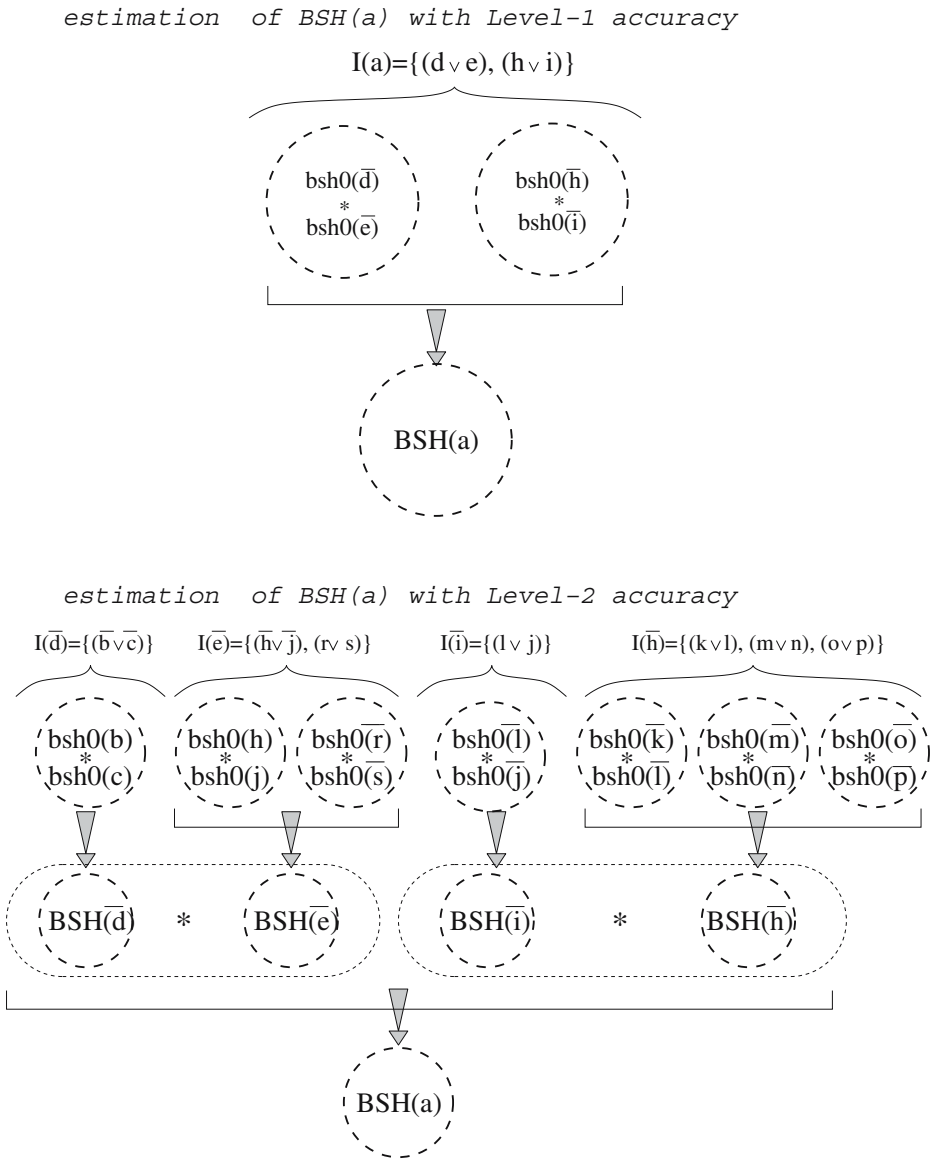

Figure 6 Estimation of $BSH(a)$ on the set of clauses in Figure 5, with different levels of accuracy.

consists of the sum of the products computed in all sectors of the right-hand side circle. Finally, if t must be *true* in all assignments satisfying the maximum number of clauses (possibly all) of the subformula at the current node where $BSH(t)$ is evaluated, the correlation that $BSH(t)$ intends to estimate must be the strongest, and then $BSH(t)$ should have one of the highest values among all values which it can take at the node under consideration. In this sense, $BSH(t)$ tends to find the variables belonging to the local backbone of the subformula at the node being considered. The

chosen branching variable t is the one that has the highest score $BSH(t) \times BSH(\bar{t})$. In other words, this final product tends to select for branching, the variable that has the most chances of belonging to a local backbone of the subformula under its positive and negative signs. This has an intuitive (and practical) consequence. It tends to maximize the chances of producing unary clauses where t and $\bar{t}$ appear in a refutation subtree with a small mean depth. Then, backtracking is activated sooner than with usual branching heuristics.

2.1. BSH vs MOMS Heuristic

The aim of this section is to give an accurate idea of the impact of the BSH heuristic implemented alone in a basic DPLL procedure. For this, we carried out experimental comparisons with the well-known MOMS heuristic, which forms the basis of the heuristics of complete solvers developed to solve random SAT formulae. The DPLL procedure that we implemented corresponds to the one described in [7], namely: Let $\mathcal{F}$ be a SAT formula,

DPLL($\mathcal{F}$)

1) If $\mathcal{F}$ contains at least one empty clause, then return UNSAT; if $\mathcal{F}$ is empty, then return SAT.
2) Assign the truth value *true* to pure literals or literals of unit clauses and consequently reduce the formula $\mathcal{F}$. Iterate this operation while it is possible.
3) Select a branching variable x according to the heuristic MOMS/BSH.
4) DPLL($\mathcal{F} \wedge x$); if UNSAT then DPLL($\mathcal{F} \wedge \bar{x}$).

Regarding the MOMS heuristic, let us recall that it selects, among the variables not yet assigned a truth value, the one having the most occurrences in the shortest clauses. In the framework of our comparative experiments, in order that the comparison is as accurate and as rigorous as possible, we have implemented a modified MOMS heuristic, more efficient than the basic version. Indeed, the computation of our BSH heuristic includes a kind of look-ahead treatment. For this reason, in the implementation of MOMS, we added a look-ahead treatment that simulates what is done in the computation of BSH. This additional treatment in MOMS consists, for each literal under evaluation, in carrying out all possible unit propagations from setting the evaluated literal to true. This treatment can detect early contradictions, therefore leading to earlier backtracks, or can detect variables with a fixed truth value. These experimental conditions guarantee as much as possible that the comparison of results is actually based on the strategies of choice of branching variables by MOMS and BSH. The experiments were done on random 3-SAT formulae located around the satisfiability threshold, having 100, 200, 300, and 400 variables and 430, 850, 1,275, and 1,700 clauses. The computer used was equipped with an Intel Pentium IV, 2.6 GHz processor and ran under a Linux operating system. The mean numbers of nodes of search trees and the mean computing times obtained for MOMS and BSH are given in Tables I and II, respectively.

The mean numbers of nodes in Table I for SAT as well as UNSAT formulae are clearly in favor of BSH. Thus the sizes of trees developed by MOMS can be up to 14 times larger than with BSH on UNSAT formulae with 400 variables. Moreover, we note that for each formula, the size of the search tree developed by BSH is always much smaller than with MOMS. The gain factor with BSH in terms of tree size increases as the number of variables of the formulae increases.

Table I Mean numbers of nodes developed by DPLL with the MOMS and BSH branching heuristics on 100 random formulae with 100, 200, 300, and 400 variables around the satisfiability threshold.

No. of variables	No. of clauses	Sat/Unsat	MOMS mean nodes	BSH mean nodes
100	430	Unsat	32	19
		Sat	17	13
200	850	Unsat	1, 719	548
		Sat	546	199
300	1, 275	Unsat	78, 721	12, 292
		Sat	20, 341	3, 664
400	1, 700	Unsat	4, 252, 782	303, 448
		Sat	827, 606	87, 650

Regarding the mean computing times given in Table II, comparisons turn out in favor of BSH, as in the case of tree size. Note, in particular, that under the conditions of these experiments, the BSH heuristic has been applied all along the development of the search tree, which is very time consuming and so penalizes the global efficiency of BSH. That is the reason that in *cnfs* or *kcnfs*, *BSH* is applied only within the top part of the tree. In spite of this penalty, the computing time with *BSH* can be eight times smaller than with MOMS on unsatisfiable formulae with 400 variables. As for the size of search trees, it can also be observed that the gain factor in computing time with *BSH* increases as the number of variables increases. Moreover the experiments have shown that limiting the use of *BSH* to the choice of a branching variable where clauses of length 3 are involved, and stopping its use as soon as this choice depends on clauses of length 2, the mean solving time for unsatisfiable formulae with 400 variables already goes down to 122 s, which is better than any solver of Figure 8. For example, for *satz* in its complete version the running time is 145 s. All in all, these experimental results show clearly and with no ambiguity that *BSH* carries out a selection of branching variable that is superior to the classical strategies.

3. *k*-SAT Formulae

The measure of correlation of the literal t with other variables of a subformula, computed by $BSH(t)$, depends, on the one hand, on the length of the clause under

Table II Mean computing times of DPLL with MOMS and BSH branching heuristics on 100 random formulae with 100, 200, 300, and 400 variables around the satisfiability threshold.

No. of variables	No. of clauses	Sat/Unsat	MOMS mean time	BSH mean time
100	430	Unsat	0.01	0.01
		Sat	0.01	0.01
200	850	Unsat	0.59	0.33
		Sat	0.19	0.12
300	1,275	Unsat	43.19	11.18
		Sat	11.24	3.56
400	1,700	Unsat	3, 291.04	416.33
		Sat	640.32	120.60

consideration where t appears and, on the other hand, on the correlations of the other literals themselves associated to t in this clause. Let us give the intuitive meaning of these two factors of correlation. For the first factor, that is, the length of clauses, if t occurs in the following clause of length 6, $(t \vee u_1 \vee v_1 \vee w_1 \vee x_1 \vee y_1)$, its correlation with the rest of the variables through $(u_1 \vee v_1 \vee w_1 \vee x_1 \vee y_1)$ is likely weaker in a random formula than if it occurs in the following clause of length 3 : $(t \vee u_2 \vee v_2)$ through $(u_2 \vee v_2)$. As an extreme example, if t is a unit clause in a formula, t must be set to *true* with any truth assignment to the other variables in order to satisfy the maximum number of clauses of the formula. For the second factor, that is, the correlations of other literals of the clause where t appears, if in the above clause, u_2 and v_2 are weakly correlated with the other variables of the formula, for example if they are pure literals (which is an extreme case), then through a clause of any length, however large, where t occurs, t is more correlated with the rest of the variables than through $(u_2 \vee v_2)$. This latter factor is naturally taken into account by the level of evaluation defined for $BSH(t)$ in Figure 4, Section 2, whose principle extends directly to clauses with various lengths (see also Figure 7). Contrary to the principle described in the previous section, where the computation of $BSH(t)$ on a 3SAT formula (possibly simplified by a partial truth assignment to the variables) is based on the set $\mathcal{I}(t)$ consisting solely of clauses of length 2, the computation of $BSH(t)$ on a formula with clauses of various lengths is based on a set $\mathcal{I}(t)$, which also contains clauses of various lengths. This can bring about disparities in estimating the correlations of distinct occurrences of t with the rest of the variables. For example,

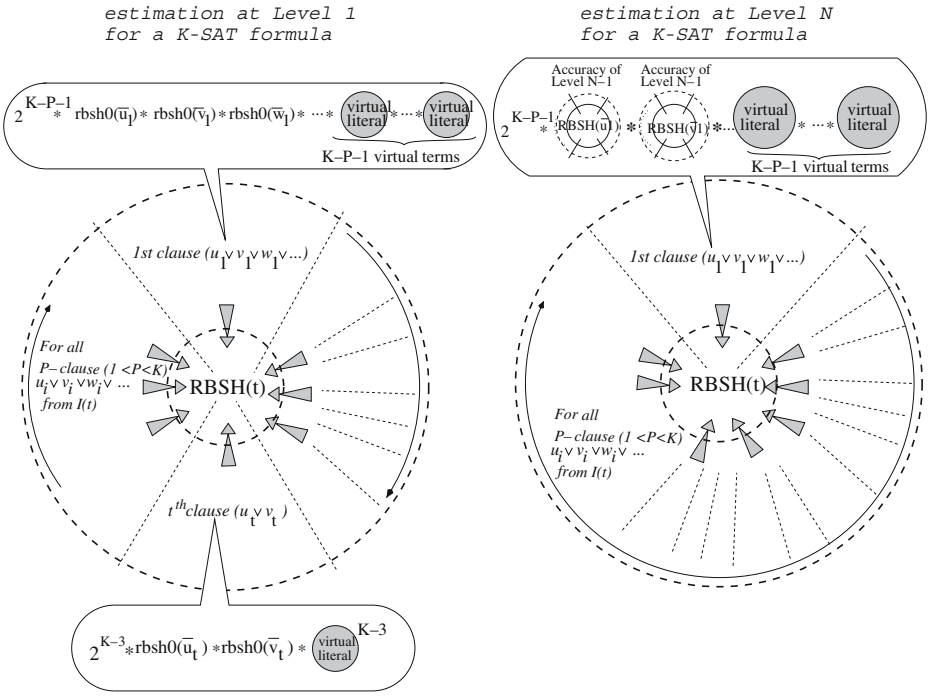

Figure 7 Function $RBSH(t)$ of the kSAT branching variable selection heuristic.

if two occurrences of t can be forced to *true* through two clauses of different lengths, say z_1 and z_2, with $z_2 \geq z_1$, a direct generalization of the computation of $BSH(t)$ in Section 2 could give a larger number of possibilities that t is forced to *true* in the clause having the largest length, z_2, than in the clause with length z_1. This seeming incoherence does not mean that the principle of computation described in Section 2 is not sound. The reason is merely that the correlations are not estimated at the same scale and therefore must be normalized. As indicated in Figure 7, this normalization consists in introducing a 'virtual' literal with which we fill the clauses in the set $\mathcal{I}(t)$ so that they all have the same length, equal to $K-1$ where K is the largest possible clause length in the subformula associated with the current node. A reasonable value of $RBSH$ must be associated with this virtual literal, so that the evaluation of the correlation of t makes sense. We choose to assign it, for a given level of accuracy, the mean of all values computed for the literals belonging to clauses in the sets $\mathcal{I}$ associated with a literal under evaluation at a current node of the search tree. Finally, since the literal is virtual, we can choose adequately its truth value as *false*. Hence there is no uncertainty about its truth value; the probability of it being *false* is 1, whereas the probability for 'real' literals to be *false* or *true* can be roughly estimated (in a random assignment) at $\frac{1}{2}$. Consequently, each time the virtual literal is used, its value is weighted by a factor 2 (see the term 2^{K-P-1} of the products associated to each sector of the circles of Figure 7). Thus we obtain a complete normalization in the computation of the correlation of t with the rest of variables. The normalization is to be done at every node of the solving tree and is specific for each subformula associated with the node under consideration. That is what we mean when we say that we have to renormalize in the course of the development of the solving tree.

Following the description we have made for BSH in Figure 4, the Renormalized Backbone-Search Heuristic $RBSH$ is described in Figure 7. The computation carried out by $RBSH$ is as follows. As with BSH, $RBSH$ is the result of a computation based on the set $\mathcal{I}(t)$. t is any literal not assigned a value at a current node of the solving tree, and $RBSH(t)$ is the score of t. K is the maximum length of the clauses in the sub-formula associated with the node under consideration. For clauses of the set $\mathcal{I}(t)$, any of which being *false* forces t to be *true* unless a contradiction occurs, P is the length of the clause being considered. P varies from 2 (using the principle of the chains of unary clauses described in Section 4) up to at most $K-1$. As previously for BSH, $RBSH$ can be evaluated at different levels of accuracy using a recursive computation (see the right-hand side circle of Figure 7). The higher the level of recursive computation, the more accurate the measure is considered to be. At level 1 (see the left-hand side circle of Figure 7), this measure is given directly by the sum of the values computed in the sectors of Figure 7. The function $rbsh0(t)$ is a generalization of $bsh0(t)$ in Figure 4, described in the previous section. Thus $rbsh0(t)$ is defined as $\sum_{j\in\{2,\sim,\ldots,K-1\}} p_j(t) \times 2^{k-j}$, where $p_j(t)$ is the number of occurrences of t in the clauses of length j. Each sector in Figure 7 represents the weighted number of possibilities that a clause of length P of $\mathcal{I}(t)$ is *false*. Finally, the branching variable chosen to be assigned successively *true* and *false* at a current node of the solving tree is one of the variables having the highest score $RBSH(t) \times RBSH(\bar{t})$. Let us take an example showing specifically how the renormalization is done. Consider that x appears in the two following clauses: $\{(x \vee \bar{a} \vee b \vee d), (x \vee c \vee e)\}$; 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, namely $rbsh0(\alpha)$, is the mean $(rbsh0(\overline{a}) + rbsh0(b) + rbsh0(d) + rbsh0(e))/4$.

4. Experimental Results and Conclusion

The solver *kcnfs* is the DPLL implementation we have first proposed in [11]. It integrates *BSH* and *RBSH* into its branching variable selection heuristic. To date, *kcnfs* remains the most efficient and robust solver for the purpose of determining whether a random *k*SAT formula is unsatisfiable for arbitrary *k*.

Figures 8 and 9 provide comparative results in terms of mean computation times between *kcnfs* and the solvers *csat* [9], *posit* [16], *satz* [24, 25], *OkSolver* [21] and *zchaff* [29, 36]. These solvers are the best-known complete solvers for solving random *k*SAT formulae. We have compared the performance of these solvers with *kcnfs* on proving contradiction of unsatisfiable random 3SAT, 4SAT, 6SAT, and 8SAT formulae where the ratio $\frac{\#clauses}{\#vars}$ is set around the satisfiability threshold (i.e., respectively equal to 4.25, 9.66, 44.2 and 180 [10, 15]).

Table III provides, for all solvers, the mean sizes of the refutation trees developed, in terms of numbers of nodes, numbers of branches, or numbers of leaves. First, these comparative results confirm that *satz* is designed especially to solve random 3SAT, for which it yields good performance. The performance of *satz* decreases significantly as *k* increases. For example, it can be observed in the comparison chart of mean computing times on hard random unsatisfiable 8SAT formulae (Figure 9) that *satz* is, on average, over twice slower than *posit*. It appears that *posit*, which is the more generic implementation of DPLL (with *csat*), remains for large *k* the best competitor of *kcnfs*. *zchaff* was not able to respond in a reasonable time (less than 3 h). The experiments involved random *k*SAT formulae up to $k = 8$. *kcnfs* exhibits each time the best performance, such that the ratio $\frac{mean\ time\ of\ the\ best\ competitor}{mean\ time\ of\ kcnfs}$ increases as *k* increases. This observation shows that the advantage of *kcnfs* is mainly due to its branching heuristic. Indeed, the more *k* increases, the fewer unit propagations and various local treatments such as look-ahead, picking, and pickback [11], are executed

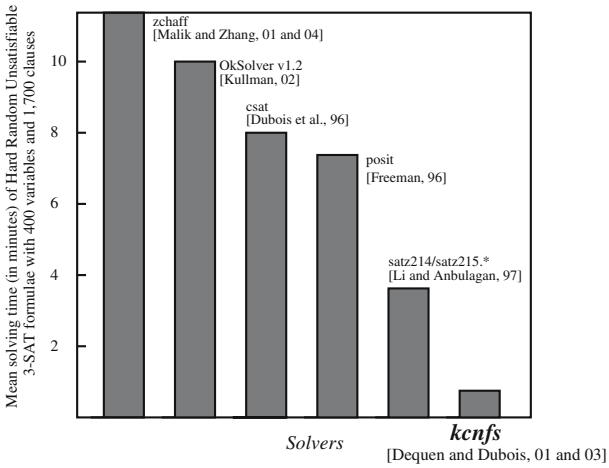

Figure 8 Comparison of mean solving times between the best of the main specialized solvers on a set of 100 unsatisfiable hard random formulae with 400 variables and 1,700 clauses.

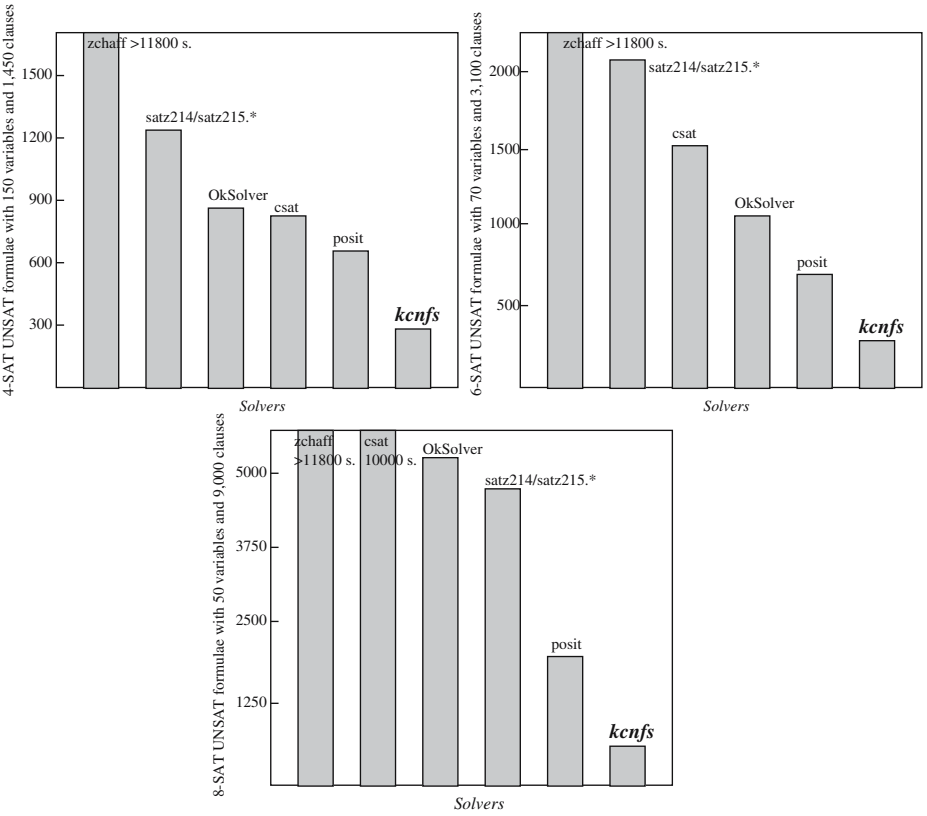

Figure 9 Comparison of mean solving times between the best of the main specialized solvers on sets of 100 unsatisfiable hard random kSAT formulae for k equal to 4, 6, and 8, with 150, 70, and 50 variables, respectively, and with 1,450, 3,100, and 9,000 clauses, respectively.

Table III Comparison of mean sizes of search trees between the best of the main specialized solvers on sets of unsatisfiable hard random kSAT formulae for k equal to 3, 4, 6, and 8 with 400, 150, 70, and 50 variables, respectively, around the threshold of satisfiability.

	csat *in #nodes* (*std. dev.*)	posit *in #nodes* (*std. dev.*)	satz *in #nodes* (*std. dev.*)	OkSolver *in #nodes* (*std. dev.*)	kcnfs *in #nodes* (*std. dev.*)
3SAT, 400v, 1,700c	2.8×10^6	3.9×10^6	0.7×10^6	1.0×10^6	0.2×10^6
4SAT, 150v, 1,450c	8.6×10^6	7.7×10^6	4.3×10^6	3.4×10^6	1.9×10^6
6SAT, 70v, 3,100c	5.2×10^6	3.4×10^6	6.8×10^6	2.9×10^6	1.4×10^6
8SAT, 50v, 9,000c	5.8×10^6	3.3×10^6	7.0×10^6	3.5×10^6	1.9×10^6

in solving a formula. Hence, for large k the performance is mainly due to the branching choice heuristic. Finally, these experimental results clearly show the progress achieved by the community over the last decade with regard to the resolution of random problems.

All comparative experiments were made on a 1 GHz Linux PC. To date, *kcnfs* remains the only solver able to answer the first challenge of Selman and Kautz [31], initially proposed in [33]. Moreover, *kcnfs* won the international $SAT'2003$ [22], $SAT'2004$ [23] competitions in the category of UNSAT random formulae with 34 and 55 competitors, respectively, and won $SAT'2005$ in the whole random category, SAT and UNSAT with 43 competitors. The detailed results of the $SAT'2003$ competition[1] show that *kcnfs* solved all the UNSAT formulae solved by at least one other competitor and that it solved formulae that were solved by none of the other solvers. During the $SAT'2004$ competition[2] *kcnfs* remained the best solver in the random UNSAT category. Moreover, for all random formula categories, *kcnfs* solved the most new problems since the $SAT'2003$ competition.[2] These results confirm that *kcnfs*, because of its branching variable selection heuristic, has a robust approach to proving the unsatisfiability of random formulae.

Acknowledgments We thank the anonymous reviewers for their very useful comments, which helped to improve the readability of the paper, and we are particularly grateful to one of them for his very careful suggestions.

References

1. Bailleux, O., Boufkhad, Y.: Efficient CNF encoding of Boolean cardinality constraints. In: Principles and Practice of Constraint Programming–CP2003: 9th Internatioanl Conference, LNCS, vol. 2833, pp. 108–122 (2003)
2. Beame, P., Karp, R., Pitassi, T., Saks, M.: The efficiency of resolution and Davis–Putnam procedures. SIAM **31**(4), 1048–1075 (2002)
3. Boufkhad, Y., Dubois, O.: Length of prime implicants and number of solutions of random CNF formulae. Theor. Comp. Sci. **215**(1–2), 1–30 (1999)
4. Braunstein, A., Mezard, M., Zecchina, R.: Survey propagation: An algorithm for satisfiability. Random Struct. Algorithms **27**, 201–206 (2005)
5. Cocco, S., Monasson, R.: Heuristic average-case analysis of the backtrack resolution of random 3-satisfiability instances. Theor. Comp. Sci. **320**(2–3), 345–372 (2004)
6. Crawford, J.M., Auton, L.D.: Experimental results on the crossover point in satisfiability problems. In: Proceedings of the 11th National Conference on Artificial Intelligence, pp. 21–27 (1993)
7. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. J. Assoc. Comput. Mach. **5**, 394–397 (1962)
8. Dequen, G., Dubois, O.: kcnfs: An efficient solver for random k-SAT formulae. In: International Conference on Theory and Applications of Satisfiability Testing (SAT), Selected Revised Papers, LNCS, vol. 6, pp. 486–501 (2003)

[1] http://www.satcompetition.org

[2] http://www.lri.fr/~simon/contest/results/

9. Dubois, O., Andre, P., Boufkhad, Y., Carlier, J.: SAT versus UNSAT. In: DIMACS Series in Discr. Math. and Theor. Computer Science, pp. 415–436 (1993)
10. Dubois, O., Boufkhad, Y.: A general upper bound for the satisfiability threshold of random *r*-SAT formulae. J. Algorithms **24**(2), 395–420 (1997)
11. Dubois, O., Dequen, G.: A backbone search heuristic for efficient solving of hard 3-SAT Formulae. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence. Seattle, pp. 248–253 (2001a)
12. Dubois, O., Dequen, G.: The non-existence of (3,1,2)-conjugate orthogonal idempotent Latin square of order 10. In: Proceedings of CP'2001, pp 108–120 (2001b)
13. Dubois, O., Mandler, J.: The 3-XORSAT threshold. In: Proceedings of the 43rd Symposium on Foundations of Computer Science, pp. 769–778 (2002)
14. Feige, U.: Relations between average case complexity and approximation complexity. In: Proceedings of the Thirty-fourth Annual ACM Symposium on Theory of Computing, Montreal, Quebec, Canada, pp. 534–543 (2002)
15. Freeman, J.W.: Improvements to propositional satisfiability search Algorithms. Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia (1995)
16. Freeman, J.W.: Hard random 3-SAT problems and the Davis–Putnam procedure. Artif. Intell. **81**(1–2), 183–198 (1996)
17. Goerdt, A., Jurdzinski, T.: Some results on random unsatisfiable k-SAT instances and approximation algorithms applied to random structures. Comb. Probab. Comput. **12**(3), 245–267 (2003)
18. Grégoire, E., Ostrowski, R., Mazure, B., Sais, L.: Automatic extraction of functional dependencies. In: Testing: 7th International Conference (SAT'04), Revised Selected Papers, LNCS, vol. 3542, Springer, pp. 122–132 (2005)
19. Hoos, H.H.: An adaptive noise mechanism for walkSAT. In: Eighteenth National Conference on Artificial Intelligence, pp. 655–660 (2002)
20. Hoos, H.H., Stutzle, T.: Local search algorithms for SAT: An empirical evaluation. J. Autom. Reason. **24**(4), 421–481 (2000)
21. Kullman, O.: Heuristics for SAT algorithms: A systematic study. In: Extended abstract for the Second Workshop on the Satisfiability Problem (SAT'98) (1998)
22. Leberre, D., Simon, L.: The essentials of the SAT 2003 competition. In: International Conference on Theory and Applications of Satisfiability Testing (SAT), Revised Selected Papers, LNCS, vol. 6 (2003)
23. Leberre, D., Simon, L.: Fifty-five solvers in Vancouver: The SAT 2004 Competition. In: Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing, SAT 2004, Revised Selected Papers, LNCS, vol. 3542, Springer, pp 321–344 (2005)
24. Li, C.M.: Exploiting yet more the power of unit clause propagation to solve the 3-SAT problem. In: Proceedings of European Conference on Artificial Intelligence. pp. 11–16 (1996)
25. Li, C.M., Anbulagan: Heuristics based on unit propagation for satisfiability problems. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence. Nagoya, Japan, pp. 366–371 (1997a)
26. Li, C.M., Anbulagan: Look-ahead versus look-back for satisfiability problems. In: Lecture Notes in Computer Science 1330, pp 341–355 (1997b)
27. Mezard, M., Parisi, G., Zecchina, R.: Analytic and algorithmic solutions of random satisfiability problems. Science **297**, 812–815 (2002)
28. Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., Troyansky, L.: 2+p-SAT: Relation of typical-case complexity to the nature of the phase transition. RSA: Random Struct. Algorithms **15**, 414–440 (1999)
29. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of 9th Design Automation Conference. Las Vegas (2001)
30. Rosenthal, J.W., Plotkin, J.W., Franco, J.: The probability of pure literals. J. Log. Comput. **9**(4), 501–513 (1999)
31. Selman, B., Kautz, H.A.: Ten challenges redux: Recent progress in propositional reasoning and search. Invited paper, Ninth International Conference on Principles and Practice of Constraint Programming (CP 2003). Cork (Ireland) (2003)
32. Selman, B., Kautz, H.A., Cohen, B.: Noise strategies for improving local search. In: Proceedings of the 12th National Conference on Artificial Intelligence, vol. 1, pp. 337–343. Menlo Park, California (1994)

33. Selman, B., Kautz, H.A., McAllester, D.A.: Ten challenges in propositional reasoning and search. In: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97), pp. 50–54 (1997)
34. Singer, J., Gent, I., Smail, A.: Backbone fragility and the local search cost peak. J. Artif. Intell. Res. **12**, 235–270 (2000)
35. Slaney, J., Walsh, T.: Backbones in optimization and approximation. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence. Seattle, pp. 254–259 (2001)
36. Zhang, L., Madigan, C., Moskewicz, M., Malik, S.: Efficient conflict driven learning in a boolean satisfiability solver. In: Proceedings of ICCAD, 279–285, IEEE Press, Piscataway, NJ