Available online at www.sciencedirect.com
ScienceDirect
Discrete Applied Mathematics 155 (2007) 1613–1626
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

Regular-SAT: A many-valued approach to solving combinatorial problems

Ramón Béjar[a], Felip Manyà[b], Alba Cabiscol[a], Cèsar Fernàndez[a], Carla Gomes[c]

[a] *Department of Computer Science, Universitat de LLeida, Jaume II 69, 25001 LLeida, Spain*
[b] *IIIA, Artificial Intelligence Research Institute,CSIC, Spanish Council for Scientific Research,Campus UAB, 08193 Bellaterra, Spain*
[c] *Department of Computer Science, Cornell University, Ithaca, NY 14853, USA*

Received 21 March 2001; received in revised form 14 July 2003; accepted 19 October 2005
Available online 15 December 2006

Abstract

Regular-SAT is a constraint programming language between CSP and SAT that—by combining many of the good properties of each paradigm—offers a good compromise between performance and expressive power. Its similarity to SAT allows us to define a uniform encoding formalism, to extend existing SAT algorithms to Regular-SAT without incurring excessive overhead in terms of computational cost, and to identify phase transition phenomena in randomly generated instances. On the other hand, Regular-SAT inherits from CSP more compact and natural encodings that maintain more the structure of the original problem. Our experimental results—using a range of benchmark problems—provide evidence that Regular-SAT offers practical computational advantages for solving combinatorial problems.

Keywords: Combinatorial problem solving; Many-valued logic; Satisfiability; Solvers

1. Introduction

Boolean satisfiability (SAT) is nowadays a well-studied NP-complete problem that provides a simple, uniform and powerful formalism for representing and solving other combinatorial problems, including circuit verification [24,27,31], quasigroup completion [15], planning [22], and scheduling [8]. In recent years, the study of the search behavior of random SAT formulas has provided tremendous insights into the hardness nature of combinatorial problems—beyond the worst-case notion of NP-completeness—and has led in turn to the development of highly optimized SAT solvers that implement advanced search techniques with clever heuristics, learning, randomization and restarts. In particular, such studies have uncovered, for ensembles of random instances, an interesting phase transition behavior between an area in which most instances are solvable and one in which most of the instances are unsolvable [9,25]—the critically constrained area, where the hardest instances occur, coincides with the phase transition.

In parallel to the advances achieved on SAT, the community working on many-valued logics has studied a class of clausal forms—so-called regular CNF formulas—that are used as a powerful formalism for representing and solving the satisfiability problem of any finite-valued logic. By using a generic clausal form translation algorithm [18], a formula

E-mail addresses: ramon@diei.udl.es (R. Béjar), felip@iiia.csic.es (F. Manyà), alba@diei.udl.es (A. Cabiscol), cesar@diei.udl.es (C. Fernàndez), gomes@cs.cornell.edu (C. Gomes).

0166-218X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2005.10.020

A from any finitely valued logic can be translated into a satisfiability equivalent regular CNF formula B in polynomial time. So, deciding whether A is satisfiable is reduced to deciding whether B is satisfiable. This way is not necessary to develop a particular resolution calculus for each finitely valued logic. Any resolution calculus for regular CNF formulas becomes a resolution calculus for any finitely valued logic [3,19].

Regular CNF formulas are similar to Boolean CNF formulas, except that their domain is now totally ordered and not limited to two truth values, and use a generalized notion of literal. Given a domain (truth value set) T ($|T| \geqslant 2$) equipped with a total ordering $\leqslant$, a *regular literal* is an expression of the form $S : p$, where p is a propositional variable and S is a subset of T which is either of the form $\uparrow i = \{j \in T | j \geqslant i\}$ or of the form $\downarrow i = \{j \in T | j \leqslant i\}$ for some $i \in T$. The informal meaning of $S : p$ is "p is constrained to the values in S". Regular-SAT is the problem of deciding the satisfiability of regular CNF formulas.

The success of SAT as a generic problem solving approach, as well as the similarities between SAT and Regular-SAT at the logical and complexity levels, led us to investigate Regular-SAT as a constraint programming language between CSP and SAT that, by combining many of the good properties of each paradigm, offers a good compromise between performance and expressive power. As we show in the rest of the paper, its similarity to SAT allows us to define a uniform encoding formalism, to extend existing SAT algorithms to Regular-SAT without incurring excessive overhead in terms of computational cost, and to identify phase transition phenomena in randomly generated instances. On the other hand, Regular-SAT inherits from CSP more compact and natural encodings that maintain more the structure of the original problem. Our experimental results—using a range of benchmark problems—provide evidence that Regular-SAT offers practical computational advantages for solving combinatorial problems.

The paper is structured as follows. We begin by formally defining the satisfiability problem of Regular CNF formulas (Section 2). In Section 3 we discuss the advantages of Regular-SAT encodings. In Section 4 we describe Regular-DP and Regular-WalkSAT, which are generalizations of the so-called Davis–Putnam procedure (though it is actually due to Davis et al. [11]) and WalkSAT [29]. In Section 5, we present a detailed evaluation of the performance of Regular-SAT procedures on several combinatorial problems, we compare Boolean SAT and Regular SAT w.r.t. capturing problem structure, and we describe the phase transition phenomena we identified for Regular Random 3-SAT. Section 6 gives overall conclusions.

2. Regular CNF formulas

Definition 1. A *truth value set* is a non-empty set $T = \{i_1, i_2, \ldots, i_n\}$, equipped with a total ordering $\leqslant$. A *sign* is a set $S \subseteq T$ of truth values. For each element i of the truth value set T, let $\uparrow i$ denote the sign $\{j \in T \mid j \geqslant i\}$, and let $\downarrow i$ denote the sign $\{j \in T \mid j \leqslant i\}$. A sign S is *regular* if it is identical to $\uparrow i$ or to $\downarrow i$ for some $i \in T$.

Definition 2. A *regular literal* is an expression of the form $S : p$, where S is a regular sign and p is a propositional variable. The complementary literal of $S : p$ is $(T \setminus S) : p$. A regular literal $S : p$ is of *positive* (*negative*) *polarity* if S is of the form $\uparrow i$ ($\downarrow i$) for some $i \in T$. A *regular clause* is a finite set of regular literals. A *regular CNF formula* is a finite set of regular clauses.

Example 1. Let T be the set $\{0, 1, 2\}$ with the standard order on natural numbers. An example of regular CNF formula is

$$(\downarrow 0 : p_1 \vee \downarrow 1 : p_2 \vee \uparrow 2 : p_3) \wedge (\uparrow 1 : p_1 \vee \downarrow 0 : p_2).$$

Definition 3. An *interpretation* is a mapping that assigns to every propositional variable an element of the truth value set. An interpretation I satisfies a regular literal $S : p$ iff $I(p) \in S$. An interpretation satisfies a regular clause iff it satisfies at least one of its regular literals. A regular CNF formula Γ is *satisfiable* iff there exists at least one interpretation that satisfies all the regular clauses in Γ. A regular CNF formula that is not satisfiable is *unsatisfiable*. The empty regular clause, denoted by $\square$, is always unsatisfiable and the empty regular CNF formula is always satisfiable.

3. Regular-SAT encodings

One advantage of Regular-SAT w.r.t. to SAT is that it produces more compact and natural encodings of combinatorial problems. As now the domain of variables is not restricted to two elements, we need fewer variables and fewer clauses,

and the intended meaning of each variable becomes much more clear to the user. Let us illustrate these facts with the quasigroup domain. The problem of constructing a quasigroup consists of coloring the n^2 cells of an $n \cdot n$ matrix, with n colors, such that there are no repetitions of color in each row and each column.

In the SAT encoding, each variable represents a color assigned to a particular cell, so if n is the order of the quasigroup, we have n^3 variables (n^2 cells with n colors each). Then, we generate clauses that encode the following constraints:

(1) Some color must be assigned to each cell.
(2) No color is assigned to two cells in the same row.
(3) No color is assigned to two cells in the same column.

The first constraint generates clauses of length n with positive literals, and the second and third ones generate binary clauses with negative literals. The total number of clauses generated is $O(n^4)$.

In the Regular-SAT encoding, each variable represents a cell of the quasigroup and the truth value assigned to it represents the color of the cell, so we have n^2 variables and n truth values. Then, we generate clauses that encode the same constraints as in the SAT encoding, except for the first constraint. This constraint does not need to be stated explicitly in the Regular-SAT encoding, because a many-valued interpretation to the variables of the formula ensures that each cell receives exactly one color. For encoding the constraint that a particular color i cannot be assigned to two different cells $c1$ and $c2$ of the same row (or column) we generate a regular clause of the form

$$\downarrow i-1 : c_1 \vee \uparrow i+1 : c_1 \vee \downarrow i-1 : c_2 \vee \uparrow i+1 : c_2.$$

By repeating this clause for all the possible colors, we ensure that $c1$ and $c2$ do not receive the same color.

4. Regular-SAT solvers

We have designed and implemented two solvers for Regular-SAT: Regular-DP—a complete solver that builds on the Davis–Putnam procedure [11]—and Regular-WalkSAT—an incomplete solver that builds on WalkSAT [29].

Regular-DP generalizes the one-literal rule and the branching rule as follows:

Regular one-literal rule: Given a regular CNF formula Γ containing a regular unit clause $\{S : p\}$,

(1) remove all clauses containing a literal subsumed by $\{S : p\}$; i.e., all clauses containing a literal $S' : p$ such that $S \subseteq S'$;
(2) delete all occurrences of literals $S'' : p$ such that $S \cap S'' = \emptyset$.

Regular branching rule: Reduce the problem of determining whether a regular CNF formula Γ is satisfiable to the problem of determining whether $\Gamma \cup \{S : p\}$ is satisfiable or $\Gamma \cup \{(T \backslash S) : p\}$ is satisfiable, where $S : p$ is a regular literal occurring in Γ and the regular literal $(T \backslash S) : p$ is its complement.

The pseudo-code of Regular-DP is shown in Fig. 1. It returns true (false) if the input regular CNF formula Γ is satisfiable (unsatisfiable). First, it applies repeatedly the regular one-literal rule and derives a simplified formula Γ'.

```
procedure Regular-DP
Input: a regular CNF formula Γ
Output: true if Γ is satisfiable and false if Γ is unsatisfiable
begin
  if Γ = ∅ then return true;
  if □ ∈ Γ then return false;
  /* regular one-literal rule*/
  if Γ contains a unit clause {S′:p} then Regular-DP(Γ_{S′:p});
  let S:p be a regular literal occurring in Γ;
  /* regular branching rule */
  if Regular-DP(Γ_{S:p}) then return true;
  else return Regular-DP(Γ_{(T\S):p});
end
```

Fig. 1. The Regular-DP procedure.

```
procedure Regular-WalkSAT
Input: a regular CNF formula Γ, MaxChanges, MaxTries and ω
Output: a satisfying interpretation of Γ, if found
begin
 for i := 1 to MaxTries
   I := a randomly generated interpretation for Γ;
   for j := 1 to MaxChanges
     if I satisfies Γ then return I;
     Pick one unsatisfied clause C from Γ;
     S := { (p, k) | S′: p ∈ C, k ∈ S′ };
     (p′, k′) := select-WalkSAT( S, Γ, ω );
     I := I with the truth assignment of p′ changed to k′;
 return "no satisfying interpretation found";
end
```

Fig. 2. The Regular-WalkSAT procedure.

Once the formula cannot be further simplified, it selects a regular literal $S : p$ of Γ', applies the branching rule and solves recursively the problem of deciding whether $\Gamma' \cup \{S : p\}$ is satisfiable or $\Gamma' \cup \{(T \backslash S) : p\}$ is satisfiable. In the pseudo-code, $\Gamma_{S:p}$ denotes the formula obtained after applying the regular one-literal rule to a regular CNF formula Γ using the regular unit clause $\{S : p\}$.

Our implementation of Regular-DP incorporates two branching heuristics which are extensions of the two-sided Jeroslow–Wang rule [5,20]. Given a regular CNF formula Γ, such heuristics select a regular literal L occurring in Γ that maximizes $J(L) + J(\overline{L})$, where $J(L)$ can be defined as

$$J(L) = \sum_{\substack{\exists L' : L' \subseteq L \\ L' \in C \in \Gamma}} 2^{-|C|}, \tag{1}$$

$$J(L) = \sum_{\substack{\exists L' : L' \subseteq L \\ L' \in C \in \Gamma}} \left(\prod_{S:p \in C} \frac{|T| - |S|}{2(|T| - 1)} \right), \tag{2}$$

where $\overline{L}$ denotes the complement of literal L, $L' \subseteq L$ denotes that literal L' subsumes literal L, $|C|$ denotes the number of literals in clause C, and $|S|$ the number of truth values in sign S.

Eq. (1) assigns a larger value to those regular literals L subsumed by regular literals L' that appear in many small clauses. This way, when Regular-DP branches on $\overline{L}$, the probability of deriving new regular unit clauses is larger. Eq. (2), that was used in our experiments, takes into account the length of regular signs as well. This fact is important because regular literals with small signs have a larger probability of being eliminated during the application of the regular one-literal rule. When $|T| = 2$ we get the same equation.

Regular-WalkSAT, whose pseudo-code is shown in Fig. 2, tries to find a satisfying interpretation for a regular CNF formula Γ performing a greedily biased walk through the space of possible interpretations. It starts with a randomly generated interpretation I. If I does not satisfy Γ, it proceeds as follows: (i) it randomly chooses an unsatisfied clause C, (ii) it chooses—using function select-WalkSAT—a variable-value pair (p', k') from the set S of pairs (p, k) such that C is satisfied by the current interpretation I if the truth value that I assigns to p is changed to k, and (iii) it creates a new interpretation I' that is identical to I except that $I'(p') = k'$. Such changes are repeated until either a satisfying interpretation is found or a pre-set maximum number of changes (MaxChanges) is reached. This process is repeated as needed, up to a maximum of MaxTries times.

Function select-WalkSAT calculates, for each pair $(p, k) \in S$, the number of broken clauses; i.e. the number of clauses that are satisfied by I but that would become unsatisfied if the assignment of p is changed to k. If the minimum number of broken clauses found (u) is greater than zero then either it randomly chooses, with probability ω, a pair (p', k') from S or it randomly chooses, with probability $1 - \omega$, a pair (p', k') from those pairs for which the number of broken clauses is u. If $u = 0$, then it randomly chooses a pair from those pairs for which $u = 0$.

To our best knowledge, the first implementations of local search algorithms for non-Boolean satisfiability were Regular-GSAT [5] and Regular-WalkSAT [6]. In our experiments we used the last available version (10.0) of Regular-WalkSAT, which is faster than the previous ones. Recently, Frisch and Peugeniez [12] have considered a class of many-valued formulas where the signs of literals are singletons (monosigned CNF formulas), and have implemented an efficient local search algorithm for those formulas. Their results show that using non-Boolean satisfiability encodings and solvers is a competitive generic problem solving approach. We are now involved in a project in which we use both monosigned and regular CNF formulas to represent and solve combinatorial problems.

5. Empirical investigation

A key question regarding Regular-WalkSAT and Regular-DP is how their performance compares to standard WalkSAT and DP, as well as how they preserve certain properties such as backbone and phase transition.

This section is divided in two parts. The first part contains our experimental results on structured instance from four problem domains: pigeon hole, graph coloring, all interval series, and quasigroup completion; and analyze how the backbone structure of the quasigroup completion problem is preserved in Regular-SAT. We refer to [4] for a detailed description of the problem encodings and more detailed run time data. The second part reports our results for uniformly generated random instances. We focus on the Regular Random 3-SAT problem and identify a phase transition phenomenon.

We first provide experimental evidence that there is a concrete computational advantage to using Regular-SAT encodings and procedures on a representative sample of problem domains we considered. Then, we show that the more compact and natural Regular-SAT encodings preserve better the problem structure than SAT encodings.

5.1. Structured instances

5.1.1. Problem domains

In our experimental investigation we considered five problem domains: graph coloring and pigeon hole for measuring the performance of systematic solvers; and all interval series, graph k-coloring, and quasigroup with holes for measuring the performance of local search solvers. We considered instances of these problems which are known to be hard for Boolean satisfiability algorithms.

The graph k-coloring instances solved by systematic search were flat graphs [10], that are graph coloring instances with at least one solution with k colors, whereas the instances solved by local search were the DIMACS satisfiable graph coloring instances DSJC125.5.col (solved with $k = 17$) and DSJC250.5.col (solved with $k = 25$), which are out of reach of existing complete SAT solvers.

The pigeon hole problem with n holes, is the problem of deciding whether we can fill n holes with $n + 1$ pigeons in such a way that no hole receives more than one pigeon and every pigeon goes to some hole. This is a very well known unsatisfiable problem, and the interest of using it as a benchmark comes from the fact that any resolution-based Boolean satisfiability procedure needs exponential size proofs to show unsatisfiability [17].

The all interval series (`ais`) problem of size n consists of finding two vectors s and v, such that (i) $s = (s_1, \ldots, s_n)$ is a permutation of $\{0, 1, \ldots, n-1\}$, and (ii) $v = (|s_2 - s_1|, |s_3 - s_2|, \ldots, |s_n - s_{n-1}|)$ is a permutation of $\{1, 2, \ldots, n-1\}$. This problem was first used for generating SAT benchmark instances for local search in [21].

The quasigroup with holes (QWH) problem [1] is an NP-complete problem in which all instances are satisfiable and thus well-suited for evaluating local search methods. QWH instances are generated by first randomly generating a complete quasigroup, and then erasing some of the colors of the quasigroup (punching "holes"). QWH is the problem of obtaining a full quasigroup from a partially colored QWH instance. Moreover, the hardness of completing a QWH instance can be finely controlled by the number of holes punched. With relatively few holes, a completion is easy because the problem is highly constrained; similarly, instances with a large fraction of holes are relatively easy to solve, since the instances are under-constrained and many possible completions exists. In [1], it is shown that there is a region of very hard completion problems in between these two extremes. The hard instances arise in the vicinity of a phase transition threshold in the average size of the so-called backbone [1]. All our QWH instances are from such hard region.

5.1.2. Systematic search

For systematic search, we compared the performance of Regular-DP with the performance of DP when solving Regular-SAT and SAT encodings, respectively, of flat graph problem instances and pigeon hole problem instances.

Table 1
Results of DP and Regular-DP on flat graph k-coloring instances

k		Vertices = 100			Vertices = 150		
		SAT	Regular-SAT	Ratio	SAT	Regular-SAT	Ratio
3	Nodes	349	89	3.92	6182	597	10.35
	Secs.	0.70	0.075	9.33	19	0.80	23.75
4	Nodes	133572	156303	0.85	2457689	3861096	0.63
	Secs.	470	191	2.46	9955	5920	1.68

Table 2
Number of nodes for DP and Regular-DP on pigeon hole instances

Holes	SAT	Regular-SAT	Ratio	Holes	SAT	Regular-SAT	Ratio
4	73	71	1.03	7	23,107	11,275	2.05
5	429	339	1.26	8	205,011	56,519	3.62
6	2941	1463	2.01	9	2,027,135	549,255	3.70

When we say DP we mean our implementation of Regular-DP but working with $T = \{0, 1\}$. In order to study only the benefits of the encodings, both algorithms used the function of Eq. (2) in the branching heuristic.

Table 1 shows the mean cost needed to solve a flat graph instance with SAT and Regular-SAT, as well as the ratio between the cost for both approaches. The cost is shown in both the number of nodes and the time needed to solve the instances. The table shows results for sets of instances obtained with different values for the number of vertices and the number of colors used. For 4 colors and 150 vertices, only 10% of instances were solved with DP, and 85% of instances were solved with Regular-DP; in both cases we used a cutoff of 4 h, and the results shown correspond to the instances successfully solved by both approaches. Observe that even if the number of nodes in Regular-DP is not smaller in all the cases, the time is always smaller. The likely explanation for this phenomenon is that the number of unit propagations per node is sufficiently small to compensate for a larger number of backtrack nodes. These results indicate that Regular-DP, using our simple branching heuristic, is more effective on the Regular-SAT encoding. In fact, the information contained in the regular literals may help the heuristic to make better decisions. We expect that by incorporating more sophisticated heuristics in Regular-DP (e.g. extensions of look-ahead [23] and look-back [2] heuristics) we will extend the range and size of instances that Regular-DP can solve faster than state-of-the-art SAT solvers.

Table 2 shows the number of nodes of the search tree created by DP and Regular-DP when solving pigeon hole instances with different number of holes. The table also shows the ratio between the number of nodes for SAT and the number of nodes for Regular-SAT. The results indicate that relative size of the search tree for Regular-SAT is smaller than for SAT as we increase the number of holes. The growing of the ratio in terms of the time needed to solve the instances is very similar to the one for the number of nodes. This can be observed in Fig. 3, where we show the scaling behavior for DP and Regular-DP on the pigeon hole instances in terms of the number of nodes of the search tree and the time needed to solve them. We observe that Regular-DP also outperforms SAT in the time needed to solve the instances.

5.1.3. Local search

We solved our local search benchmark instances with Regular-WalkSAT and WalkSAT, and observed that the mean cost needed to solve any of our instances with Regular-WalkSAT is smaller than with WalkSAT. This was true in terms of both number of flips and time, although the difference in the time needed was not as significant as in the number of flips.[1]

Fig. 4 shows the mean number of flips and mean time needed to solve instances of the all interval series (`ais`)

[1] However, we cannot consider our current version of Regular-WalkSAT (10.0) as optimized as the current one of WalkSAT (35.0).

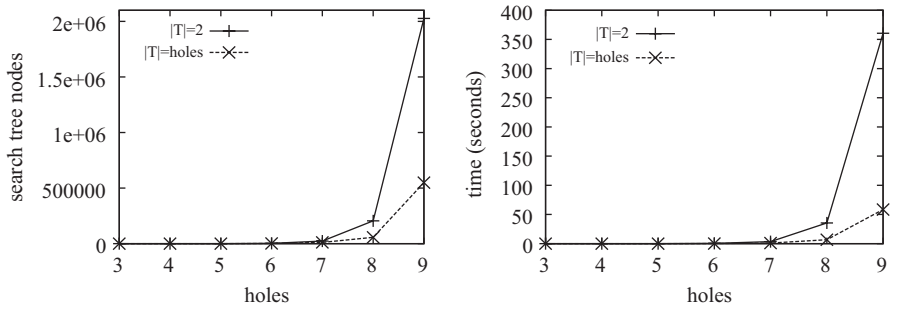

Fig. 3. Scaling behavior of DP and Regular-DP on pigeon hole instances.

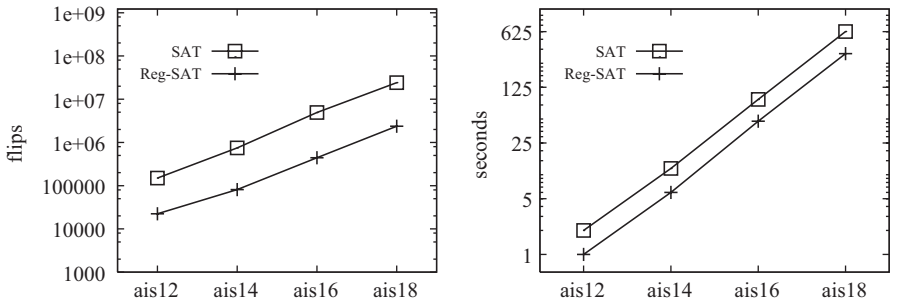

Fig. 4. Scaling behavior of Regular-SAT and SAT on the `ais` problem.

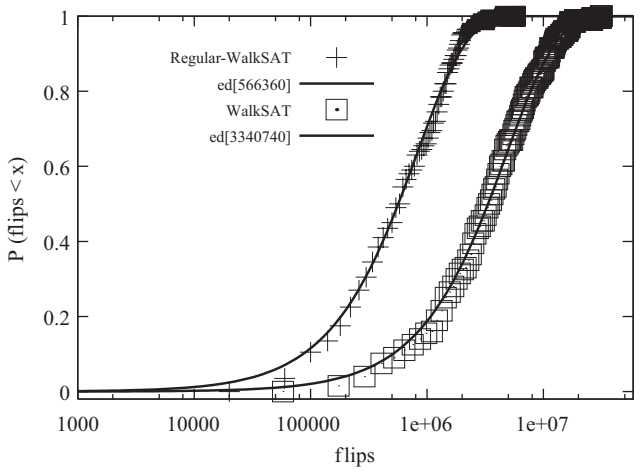

Fig. 5. RLDs for Regular-SAT and SAT on instance `DSJC125.5.col`.

problem of different size. The number of flips varies from 7 to 10 times smaller with Regular-WalkSAT and the time is always about 2 times smaller with Regular-WalkSAT.

Fig. 5 shows the cost distributions of the DIMACS graph coloring instance `DSJC125.5.col`. We observe that the cost distribution for Regular-SAT dominates the cost distribution for SAT. In other words, the probability of finding a solution in less than x flips is always greater with Regular-WalkSAT for each x. Moreover, we observed that the computational cost follows an exponential distribution (ED), at least when solving the instances with approximately optimal noise. The figure also shows the EDs that were found to best approximate the empirical distributions. The

Table 3
Median cost for SAT, Regular-SAT and CSP when solving hard QWH instances of different order (at the phase transition)

Order	Flips		Time (s)		
	SAT	Regular-SAT	SAT	Regular-SAT	CSP
27	964,849	168,455	2.1	1.5	1.7
30	2,985,105	525,884	7.2	4.8	6.9
33	11,123,065	1,520,667	27.1	16.2	57.1
36	30,972,407	5,099,701	70.8	53.9	1422.3

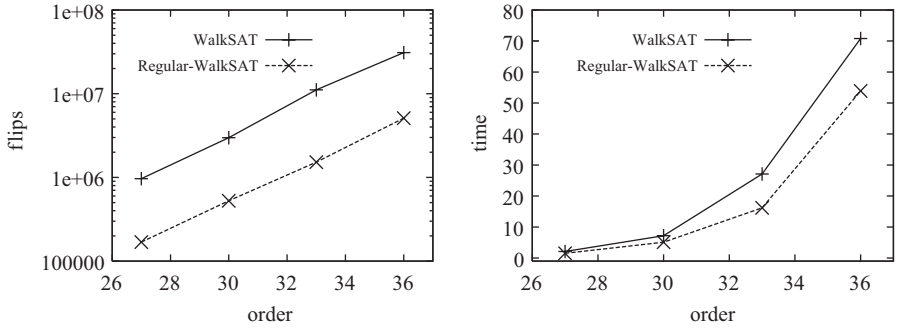

Fig. 6. Scaling behavior of the median hardness for Regular-SAT and SAT on QWH instances.

expression for the cumulative form of the EDs is $ed[m](x) = 1 - 2^{-x/m}$, where m is the median of the distribution. The approximations were derived using the Marquart–Levenberg algorithm.

For QWH, we solved hard instances of different orders. For each order, we considered 100 instances and solved the SAT and Regular-SAT encodings using WalkSAT and Regular-WalkSAT, respectively. Every instance was solved 100 times with both algorithms.

Table 3 shows the median cost, in time and flips, of all the test-sets used. The cost for a particular instance is defined as the mean time and mean number of flips needed to find a solution. We have also included results for the median time when using a CSP-based systematic search algorithm implemented with the constraint programming library ILOG and that uses the all-different constraint and the *R-brelaz-R* randomized branching strategy [16,28,30]. The results show that the median cost is smaller for the Regular-SAT approach, although between SAT and Regular-SAT the difference is more significant in terms of the number of flips. The greater difference in the number of flips can be in part attributed to the fact that the Regular-SAT encoding is more compact in terms of the number of variables. However, this difference does not directly translate in an equivalent difference in overall run time because the flip rate (flips per second) in Regular-WalkSAT is lower than in WalkSAT. At least some of this difference can be attributed to a higher level of optimization of the WalkSAT code. Despite that our implementation of Regular-WalkSAT is not so optimized, Table 3 still shows that Regular-WalkSAT also outperforms the other approaches in overall run time.

Fig. 6 shows graphically the scaling behavior in time and flips when we increase the order of the QWH instances. We see that the relative good performance of Regular-SAT scales up nicely with the order of the quasigroup. These results are consistent with the experimental results obtained with the other problem domains tested in [4] and summarized in Section 5.1.2.

We have also performed a regression analysis to study the relation between the computational cost of the two different approaches for all the instances of a given test-set. This kind of analysis allows us to investigate to what extent the superior performance observed for the median instance is also observed for any randomly obtained instance within the test-set. We performed such analysis with the sets of instances of order 27, 30, 33 and 36. Fig. 7 shows the results of the regression analysis performed with the instances of order 27 and 36. A least-mean-squares (lms) regression analysis of the logarithm of the cost was performed. The figure shows the scatter plot, where each data point (x, y) represents the logarithm, in base 10, of the mean number of flips performed by Regular-WalkSAT (x value) and the same quantity

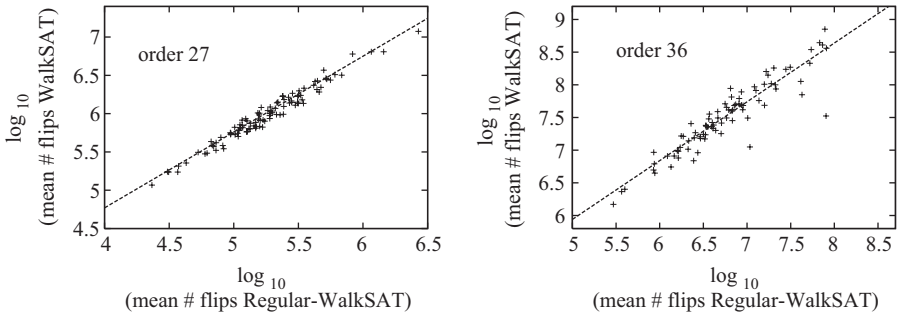

Fig. 7. Correlation between mean cost with Regular-WalkSAT and WalkSAT for QWH instances of order 27 and order 36.

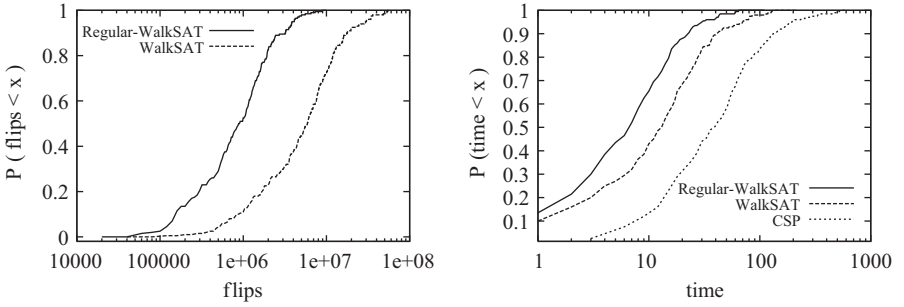

Fig. 8. RLDs (left) and RTDs (right) on the QWH median instance for order 33.

for WalkSAT (y value) when solving a particular instance. The figure also shows the linear equation obtained by the regression analysis ($\log_{10}(y) = a \log_{10}(x) + b$)[2] and the adjusted coefficient of determination (R_a^2) that quantifies to what extent the model obtained fits the experimental data. The values of R_a^2 were 0.97, 0.96, 0.91 and 0.86 for the sets of order 27, 30, 33 and 36, respectively.

The values of R_a^2 indicate that the fit of the experimental data is better for the smaller orders. A possible explanation is that as the order increases, the variability in the hardness of QWH instances increases. To properly model the correlation between instance hardness and relative performance may require a more complex regression model. Nevertheless, our analysis still suggests that the increase in performance of Regular-SAT holds fairly uniformly across each test-set.

Although the average complexity of solving instances from a problem domain distribution gives us a valuable information about the difficulty of the problem, the complexity of solving individual instances obtained with the same parameters can vary drastically from instance to instance. So, a more detailed analysis requires a study of the complexity of solving individual instances. To do so, we have constructed empirical run-time distributions (RTDs) and run-length (number of flips needed) distributions (RLDs) for both local search algorithms when solving the same instance. The methodology followed has been the one used in [21]. We have focused our attention on the median instance of a given test-set, that can be considered as the *typical* instance within a test-set. Here we present results for the test-set of quasigroups of order 33. Fig. 8 shows the RLDs and RTDs for Regular-SAT and SAT on the median instance and also the RTD for CSP on the same instance. These empirical RLDs, in the cumulative form shown, give the probability that the algorithm finds a solution for the instance in less than the number of flips of the x-axis (similarly in the RTDs). We observe that Regular-SAT strictly dominates SAT; i.e., the probability of finding a solution with Regular-SAT in

[2] Observe that by working with the logarithm of the data the actual functional relation we are fitting is $y = (10^b) \cdot x^a$.

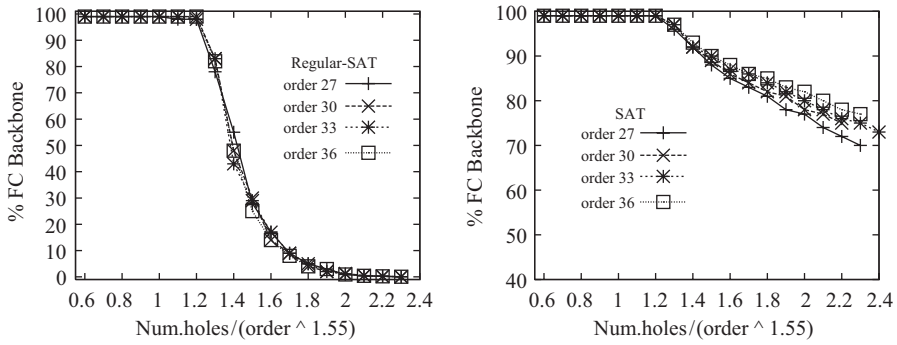

Fig. 9. The average forward-checking backbone for Regular-SAT (left) and SAT (right) on QWH instances.

less than x flips is always greater than the probability of finding a solution with SAT. Regular-SAT dominates the CSP approach even more significantly than SAT in the run time.

To show that the good results obtained with Regular-SAT are not due to implementation details, we executed the SAT instances of quasigroups considered in our experiments with Regular-WalkSAT; we took $T = \{0, 1\}$, and represented every literal p ($\neg p$) by $\uparrow 1 : p$ ($\downarrow 0 : p$). We observed that WalkSAT is slightly faster than Regular-WalkSAT on SAT encodings.

5.1.4. The backbone structure

We now consider the structure of the backbone in the QWH problem. Informally speaking, the backbone measures the amount of shared structure among the set of all solutions to a given problem instance [26]. The size of the backbone is measured in terms of the percentage of variables that have the same value in all solutions. Achlioptas et al. [1] observed a transition from a phase where the size of the backbone is almost 100% to a phase with a backbone size close to 0%. The transition is sudden and coincides with the hardest problem instances both for incomplete and complete search methods.

For efficiency purposes, Achlioptas et al. also propose a slightly weaker version of the backbone, which is computed by only using forward-checking (FC) to find shared variable settings in the solution set. They show that this backbone is qualitatively similar to the original notion of backbone. We adapted the notion of SAT FC backbone for Regular-SAT, which is obtained by applying the one-literal rule to every regular literal of the formula and computing the fraction of the total number of variables that becomes constrained to a single truth value.

The left panel of Fig. 9 shows the FC backbone for QWH instances of different orders and with a different number of holes for the Regular-SAT encoding. We observe a phase transition in the fraction of backbone variables for the Regular-SAT encoding. In contrast, the right panel of Fig. 9 displays the FC backbone structure for the Boolean SAT encoding. As we see from the figure, the SAT encoding does *not* properly preserve the phase transition properties of the backbone structure.[3] The Regular-SAT encoding can capture a structural property such as the backbone more faithfully than the Boolean SAT encoding.

5.2. Uniformly generated random instances

5.2.1. Regular random 3-SAT instances

It is well known that generating random instances of 3-SAT with the so-called uniform random model, produces instances with very different properties depending on the ratio of the number of clauses to the number of variables (C/V). Under this model, one first selects a fixed number of variables V and a fixed number of clauses C. Then, an instance is generated by selecting, uniformly at random and with replacement, every clause of the instance from the

[3] This phenomenon was initially observed by Achlioptas [1] and Gome et al. [14]. One can still recover the phase transition of the backbone for the SAT encoding by restricting the backbone count to include only the variables set positively, as done in [1].

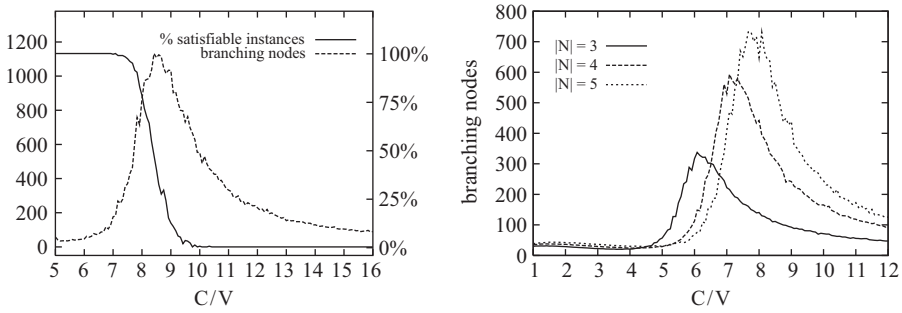

Fig. 10. Phase transition for *Regular Random 3-SAT* for $|N| = 7$ and $V = 60$ (left), and easy-hard-easy pattern for $|N| = 3, 4, 5$ and $V = 60$ (right).

possible set of three-literal non-tautological clauses one can build with the set of variables. This problem exhibits a sharp phase transition in solvability with a corresponding peak in the DP search cost as we increase C/V. The peak is approximately located around the point where 50% of the instances tend to be satisfiable. This point is called the threshold point.

The *Regular Random 3-SAT* problem, is a generalization of the classical random 3-SAT problem. In this problem we fix a number of truth values N, a number of variables V and a number of clauses C. Now the clauses are uniformly selected from the set of three-literal non-tautological regular clauses and where every regular literal has a different variable.

5.2.2. Phase transitions

In this subsection we provide experimental evidence of the existence of phase transition phenomena for *Regular Random 3-SAT*. Interestingly, we also show that the location of the threshold increases logarithmically in the cardinality of the truth value set, and give a theoretical explanation of such increase by deriving upper bounds on the location of the threshold.

Fig. 10 (left) visualizes the results of one experiment that shows the existence of phase transitions. We fixed $|N| = 7$ and $V = 60$, and generated sets of *Regular Random 3-SAT* instances in such a way that every set had a different ratio C/V. For each set of instances, the dashed line shows the average number of branching nodes per instance in the Regular-DP proof tree as a function of the ratio C/V. One can clearly observe an easy-hard-easy pattern in the computational difficulty of solving instances as the ratio C/V is varied. On the other hand, looking at the solid line—which indicates the percentage of instances that were found to be satisfiable—one can observe a sharp phase transition from satisfiable to unsatisfiable instances at a certain *threshold* of the ratio C/V. The threshold, where 50% of instances are satisfiable, corresponds to the area where the hard instances occur. Fig. 10 (right) visualizes the easy-hard-easy pattern for $|N| = 3$, $|N| = 4$ and $|N| = 5$. Observe that instances become more difficult to solve as we increase the cardinality of N, and the location of the hard region is shifted.

To investigate how the location of the threshold varies as a function of the cardinality of the truth value set, we solved sets of 60-variable $|N|$-valued *Regular Random 3-SAT* instances, for $|N| = 2$–20, 25, 30, 35, 40, 45, 50, 60, 70, with Regular-DP. From the thresholds obtained experimentally, using the Levenberg–Marquardt method for obtaining a non-linear regression model, we concluded that the location of the threshold increases logarithmically in the cardinality of the truth value set $|N|$. The equation derived was

$$L(|N|) = 6.26 \ln^{0.39}(|N|).$$

Fig. 11 (left) displays the percentage of satisfiable instances as a function of the ratio C/V for some of the cardinalities considered in our experiments. Fig. 11 (right) shows the location of the threshold as a function of $|N|$. The experimental thresholds obtained, as well as the equation derived using the Levenberg–Marquardt method, are plotted in the graph. One can observe in both figures that the increase of the location of the threshold is not linear. To provide an explanation of that fact, in the rest of this section, we derive an upper bound on the unsatisfiability threshold as a function of the cardinality of the truth value set. The derivation of such upper bounds appeared first in [7]. However, it should be noticed that one can obtain such upper bounds by using also the theory of *constrainedness* developed in [13]. Such

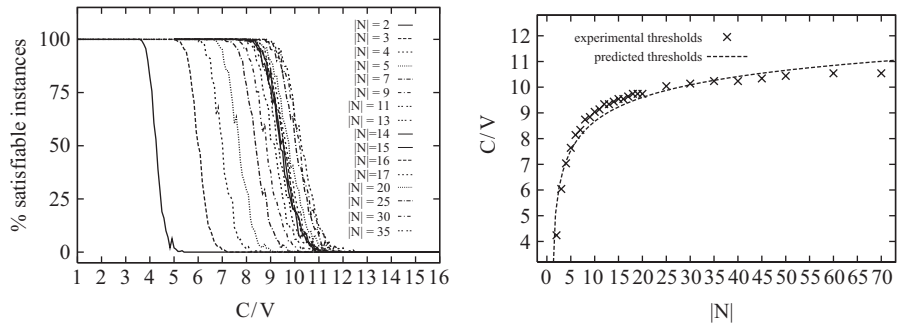

Fig. 11. Percentage of satisfiable instances as a function of C/V (left) and location of the threshold as a function of $|N|$ (right).

theory associates to a random ensemble of instances from a given NP problem a parameter that measures the relative number of expected solutions with respect to the size of the search space for an instance of the ensemble. The theory predicts that the hardest instances will be found when the expected number of solutions is 1, thus indicating we are at the threshold for unsatisfiable instances.

Let Γ be a *Regular Random 3-SAT* instance, let V be the number of propositional variables in Γ, let C be the number of regular clauses in Γ, and let $r = C/V$ be the ratio of the number of clauses to the number of variables in Γ. The problem we consider is to compute the least real number k such that if r strictly exceeds k, then the probability that Γ is satisfiable converges to 0 as V approaches infinity. We say in this case that Γ is asymptotically almost certainly unsatisfiable. A proposition stating that if r exceeds a certain constant, then Γ is asymptotically almost certainly unsatisfiable has as an immediate corollary that this constant is an upper bound for k.

We consider that the clauses of *Regular Random 3-SAT* instances are generated uniformly, independently, and with replacement. Given an interpretation I, the probability that I satisfies a regular random literal L is $\frac{1}{2}$; observe that the number of regular literals with positive polarity satisfied by an interpretation coincides with the number of regular literals with negative polarity that are not satisfied, and vice versa. The probability that I satisfies a regular random clause (with three literals) is $1-(\frac{1}{2})^3=\frac{7}{8}$. The probability that I satisfies a *Regular Random 3-SAT* instance Γ with C clauses is $(\frac{7}{8})^C$. Since there are $|N|^V$ possible interpretations for Γ, the expected number of interpretations that satisfy Γ is

$$E[|\{I|I \text{ satisfies } \Gamma\}|] = |N|^V \left(\frac{7}{8}\right)^C.$$

Since the expected number of interpretations that satisfy Γ is an upper bound on the probability that Γ is satisfiable, it holds that

$$\Pr[\Gamma \text{ is satisfiable}] \leqslant E[|\{I|I \text{ satisfies } \Gamma\}|] = |N|^V \left(\frac{7}{8}\right)^C;$$

letting $C = rV$, an upper bound for k is found by choosing r so that the expected number of interpretations that satisfy Γ converges to 0 as V approaches infinity. Thus, if $r > \log_{8/7}|N|$, then Γ is almost certainly unsatisfiable.

Therefore, if an upper bound on the unsatisfiability threshold increases logarithmically in the cardinality of the truth value set, the location of the threshold cannot increase quicker than the upper bound does. For that reason, we should get experimental thresholds increasing logarithmically or less than logarithmically in the cardinality of the truth value set.

6. Conclusions

We have shown that Regular-SAT provides an attractive approach for encoding and solving combinatorial problems. The formulation provides an intermediate alternative to the SAT and CSP approaches, and combines many of the good properties of each paradigm. Its similarity to SAT allows us to extend existing SAT algorithms to Regular-SAT without incurring excessive overhead in terms of computational cost. We have shown, using a range of benchmark problems,

that Regular-SAT offers practical computational advantages for solving combinatorial problems. In addition, Regular-SAT maintains more of the original problem structure compared to Boolean SAT encodings. By providing more powerful search heuristics and optimizing the data structures, we expect to further extend the reach of the Regular-SAT approach.

Acknowledgments

We would like to thank Bart Selman for useful comments and discussions that helped to improve the paper. Research partially supported by projects TIN2004-07933-C03-03 and TIC2003-00950 funded by the *Ministerio de Educación y Ciencia*, by the DARPA contracts F30602-00-2-0530 and F30602-00-2-0596 and by the Intelligent Information Systems Institute, Cornell University, funded by AFRL/AFOSR (F49620-01-1). The second author is supported by a grant *Ramón y Cajal*.

References

[1] D. Achlioptas, C.P. Gomes, H. Kautz, B. Selman, Generating satisfiable problem instances, in: Proceedings of AAAI-2000, 2000, pp. 256–261.
[2] R.J. Bayardo, R.C. Schrag, Using CSP look-back techniques to solve real-world SAT instances, in: Proceedings of AAAI'97, 1997, pp. 203–208.
[3] B. Beckert, R. Hähnle, F. Manyà, The SAT problem of signed CNF formulas, in: D. Basin, M. D'Agostino, D. Gabbay, S. Matthews, L. Viganò (Eds.), Labelled Deduction, Applied Logic Series, vol. 17, Kluwer, Dordrecht, 2000, pp. 61–82.
[4] R. Béjar, Systematic and local search algorithms for regular-sat, Ph.D. Thesis, Universitat Autònoma de Barcelona, 2000.
[5] R. Béjar, F. Manyà, A comparison of systematic and local search algorithms for regular CNF formulas, in: Proceedings of the Fifth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU'99, London, England, Lecture Notes in Artificial Intelligence, vol. 1638, Springer, Berlin, 1999, pp. 22–31.
[6] R. Béjar, F. Manyà, Solving combinatorial problems with regular local search algorithms, in: Proceedings of the Sixth International Conference on Logic for Programming and Automated Reasoning, LPAR'99, Tbilisi, Republic of Georgia, Lecture Notes in Artificial Intelligence, vol. 1705, Springer, Berlin, 1999, pp. 33–43.
[7] R. Béjar, F. Manyà, Phase transitions in the regular random 3-SAT problem, in: Proceedings of ISMIS'99, Lecture Notes in Artificial Intelligence, vol. 1609, 1999, pp. 292–300.
[8] R. Béjar, F. Manyà, Solving the round robin problem using propositional logic, in: Proceedings of AAAI-2000, 2000, pp. 262–266.
[9] P. Cheeseman, B. Kanefsky, W.M. Taylor, Where the really hard problems are, in: Proceedings of IJCAI-91, 1991.
[10] J. C. Culberson, F. Luo, Exploring the k-colorable landscape with iterated greedy, in: Cliques, Coloring and Satisfiability, of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 36, 1996, pp. 245–284.
[11] M. Davis, G. Logemann, D. Loveland, A machine program for theorem-proving, Commun. of the ACM 5 (1962) 394–397.
[12] A.M. Frisch, T.J. Peugniez, Solving non-boolean satisfiability problems with stochastic local search, in: Proceedings of IJCAI'01, 2001, pp. 282–288.
[13] I.P. Gent, E. MacIntyre, P. Prosser, T. Walsh, The constrainedness of search, in: Proceedings of AAAI'96, 1996.
[14] C. Gomes, H. Kautz, Y. Ruan, Qwh—a structured benchmark domain for local search, Technical Report, Intelligent Information Systems Institute (IISI), Cornell University, 2001.
[15] C.P. Gomes, B. Selman, Problem structure in the presence of perturbations, in: Proceedings of AAAI'97, 1997, pp. 221–226.
[16] C. Gomes, B. Selman, N. Crato, Heavy-tailed distributions in combinatorial search, in: Proceedings of CP'97, Lecture Notes in Computer Science, vol. 1330, Springer, Berlin, 1997, pp. 121–135.
[17] A. Haken, The intractability of resolution, Theoret. Comput. Sci. 39 (1985) 297–308.
[18] R. Hähnle, Short conjunctive normal forms in finitely-valued logics, J. Logic Comput. 4 (6) (1994) 905–927.
[19] R. Hähnle, Automated Deduction in Multiple-Valued Logics, International Series of Monographs in Computer Science, vol. 10, Oxford University Press, Oxford, 1994.
[20] R. Hähnle, Exploiting data dependencies in many-valued logics, J. Appl. Non-Classical Logics 6 (1996) 49–69.
[21] H.H. Hoos, T. Stützle, Local search algorithms for SAT: an empirical evaluation, J. Automat. Reason. 24 (4) (2000) 421–481.
[22] H.A. Kautz, B. Selman, Pushing the envelope: planning, propositional logic, and stochastic search, in: Proceedings of AAAI-96, 1996.
[23] C.M. Li, Anbulagan, Look-ahead versus look-back for satisfiability problems, in: Proceedings of CP'97, Lecture Notes in Computer Science, vol. 1330, 1997, pp. 341–355.
[24] J.P. Marques-Silva, L. Guerra, Algorithms for satisfiability in combinational circuits based on backtrack search and recursive learning, in: Proceedings of XII Symposium on Integrated Circuits and Systems Design (SBCCI), 1999.
[25] D. Mitchell, B. Selman, H. Levesque, Hard and easy distributions of SAT problems, in: Proceedings of AAAI-92, 1992.
[26] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, L. Troyansky, Determining computacional complexity from characteristic 'phase transitions', Nature 400 (8).
[27] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: engineering an efficient sat solver, in: 39th Design Automation Conference, 2001.

[28] J.-C. Régin, A filtering algorithm for constraints of difference in CSPs, in: Proceedings of AAAI'94, 1994, pp. 362–367.
[29] B. Selman, H.A. Kautz, B. Cohen, Noise strategies for improving local search, in: Proceedings of AAAI'94, 1994, pp. 337–343.
[30] K. Stergiou, T. Walsh, The difference all-difference makes, in: Proceedings of IJCAI'99, 1999.
[31] M. Velev, R. Bryant, Effective use of boolean satisfiability procedures in the formal verification of superscalar and vliw microprocessors, in: 38th Design Automation Conference (DAC'01), 2001.