Redundancy in Random SAT Formulas

Yacine Boufkhad **Olivier Roussel**
CRIL, Université d'Artois,
rue de l'Université SP 16
62307 Lens Cedex France
{boufkhad,roussel}@cril.univ-artois.fr

Abstract

The random k-SAT model is extensively used to compare satisfiability algorithms or to find the best settings for the parameters of some algorithm. Conclusions are derived from the performances measured on a large number of random instances. The size of these instances is, in general, small to get these experiments done in reasonable time. This assumes that the small size formulas have the same properties as the larger ones. We show that small size formulas have at least a characteristic that makes them relatively easier than the larger ones (beyond the increase in the size of the formulas). This characteristic is the redundancy. We show, experimentally, that the irredundant formulas are harder for both complete and incomplete methods. Besides, the randomly generated formulas tend to be naturally irredundant as their size becomes larger. Thus, irredundant small formulas are more suitable for testing algorithms because they better reflect the hardness of the larger ones.

Introduction

Random k-SAT problems are widely used to benchmark SAT algorithms. This is because the hardest instances of this class of problems are empirically well identified (Mitchell, Selman, & Levesque 1992; Larabee & Tsuji 1993). Indeed, these problems have a satisfiability phase transition behavior. Hence, as for many NP-Complete problems having the same behavior, the hardest formulas are located at the middle of this phase transition i.e. at a ratio of clauses to variables approximately equal to 4.25 for 3-SAT for example. The main interest of this class of problems is that they provide researchers working on the design of algorithms for SAT, with an inexhaustible source of hard problems to test their solving methods. Most of these algorithms, either belonging to the category of complete or incomplete methods, require the setting of one or several parameters. To find the optimal setting for these parameters, statistical methods, using trial-and-error, are generally used. The performance obtained using some parameter setting is measured statistically by running the candidate algorithm on a large set of random formulas. For these measurements to be practically feasible, the size of the instances must be kept relatively small. The best settings that are derived are generalized to larger formulas and used as the optimal ones for a candidate solving method. This generalization assumes that small size and large size formulas have the same properties and structures. We show, experimentally, that the small size formulas have at least one characteristic, beyond their size, that makes them easier for both complete and incomplete methods. This characteristic is clause *redundancy*. A clause C is said to be *redundant* in a CNF formula F, if removing C from F does not change the set of solutions of F i.e. F and $F - \{C\}$ are equivalent. A formula is said to be irredundant if none of its clauses is redundant. The hardness of random formulas at the phase transition is always implicitly evaluated with the best known algorithms, and we use the same algorithms to evaluate the hardness of formulas throughout this paper. The main contribution of this work lies in giving an empirical evidence of these two facts: irredundant formulas are harder than redundant ones and, as the number of variables increases, the formulas become less and less redundant. Indeed, we show that when generated with the usual model of k-SAT, small size formulas are highly redundant i.e. have many redundant clauses. The proportion of clauses that must be removed to make the formulas irredundant decreases rapidly and tends to 0 when the number of variables tends to infinity. Beside that fact, if redundant clauses are removed from a formula to make it irredundant then this formula becomes, in average, much more difficult for known solving methods. A straightforward consequence is the following: to design solvers that significantly increase the size of practically solved formulas one would preferably work on improving performances on irredundant small formulas.

The problem with irredundant formulas is that they are hard to generate. They require making many tests of clause redundancy which is a coNP-complete problem. In spite of that drawback one can compute and save once for all a large set of such formulas and use them for measuring the performances of a candidate algorithm. We give an algorithm that generates random irredundant formulas without requiring to test the irredundancy of all the clauses in the formula every time a new one is generated.

In this paper, we will consider randomly generated CNF formulas of fixed length clauses generated using the usual way i.e. each clause of length k is uniformly chosen at random and with replacement among the $2^k \binom{n}{k}$ possible clauses. A CNF formula is a set of clauses conjunctively interpreted. An implicate of a formula F is a clause such that every solution of F satisfies the implicate. A prime implicate is an implicate such that no proper subset of literals in this implicate is an implicate itself. An implicant of a formula is a conjunction of literals that satisfies this formula. For a clause C, we denote by $\overline{C}$ be the set of unit clauses $\{\overline{l}/l \in C\}$. We denote by C/V the ratio of the number of clauses to the number of variables. All the results reported here apply to 3SAT but some informal experiments make us believe they apply to kSAT in general.

The paper is organized as follows: in the next section the irredundant formulas are empirically shown to be harder than the redundant ones, then the number of redundant clauses is shown to tend to 0 when the number of clauses increases indefinitely and at last an algorithm for generating irredundant formulas is described.

Irredundant formulas and solving methods

We first, empirically, evaluate the hardness of the irredundant formulas with respect to the 3SAT formulas generated using the usual model. More precisely, we compare the difficulty of each formula F in a sample of randomly generated formulas with the difficulty of an irredundant subset of the clauses of F equivalent to F.

Considering the fact that a clause C is redundant in a formula F if and only if C is an implicate of $F - \{C\}$, to test the redundancy of this clause, the satisfiability of the formula $G = (F - \{C\}) \cup \overline{C}$ is tested. C is redundant in F if and only if the formula G is unsatisfiable. For a randomly generated 3CNF formula F, an equivalent irredundant formula F_{irred} is computed according to the following steps:

1. Initialize F_{irred} with F.
2. For each clause $C_i \in F_{irred}$, the satisfiability of the formula $(F_{irred} - C_i) \cup \overline{C_i}$ is tested.
3. if the latter formula is unsatisfiable then remove C_i from F_{irred}.
4. Continue with the next clause.

The resulting set of clauses F_{irred} depends on the order in which the clauses are examined. We used, merely, the chronological order in which the clauses of F are generated. Although this order is fixed, every irredundant formula equivalent to some random formula F has equal chances to be generated since F could be generated equally likely with any clause order.

On WalkSAT

We used the version of walksat described in (McAllester, Selman, & Kautz 1997) to test the hardness of irredundant formulas on local search methods. For every generated satisfiable formula F, the ratio of the performances of walksat on F_{irred} to the performances on F is computed. The same parameters setting of walksat are used to solve F and F_{irred}. As parameters, we used the Rnovelty heuristic at a noise level of 0.6 and measured the mean number of flips on 10 tries for each formula. For each formula F, we compute the ratio of the mean number of flips needed to solve F_{irred} to the mean number of flips needed to solve F. The figures 1, 2 represent the mean and the median[1] of this ratio as a function of C/V for different numbers of variables. Each point was computed using 1000 instances. Irredundant formulas prove to be harder in a range of clauses to variables ratio that depends on the number of variables and that is in the vicinity of the phase transition. For a given n, when C/V increases, the increase in the average difficulty of F_{irred} with respect to F follows the increase of the number of redundant clauses removed from F (as will be shown in the next section). It is important to be aware that even if the redundancy decreases in function of the number of variables, the ratio of mean number of flips may increase because n increases. Anyhow, this ratio is equal to 1 if F and F_{irred} are equal, which is the case when the number of variables tends to infinity. Figure 3 shows that the difficulty of solving increases as a function of the number of redundant clauses removed.

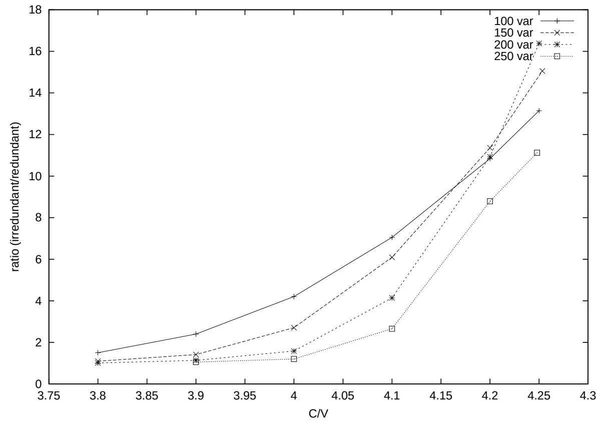

Figure 1: Median ratio (irredundant/redundant) of the number of flips in walksat as a function of C/V and for different numbers of variables

Let us stress the fact that this result does not mean that walksat fails on this type of formulas but proves that, when the parameters tuned to solve the redundant formulas are used, the irredundant ones require much more efforts to be solved. Walksat might be tuned to solve these formulas more efficiently but this would prove that experiments on small redundant formulas are not suitable to find the best parameters.

Intuitively, it is not surprising that these formulas are tricky for walksat. Indeed, the main difficulty that local search procedures have to face is that they are often stuck in local minima with few contradicted clauses. Most of the work that have been deployed to improve the performance of these procedures has consisted in finding noise strategies

[1]The median of a set of numbers is obtained by sorting the numbers and retaining the number in the middle of the list (or by averaging the two numbers in the middle of the list if it is of even length).

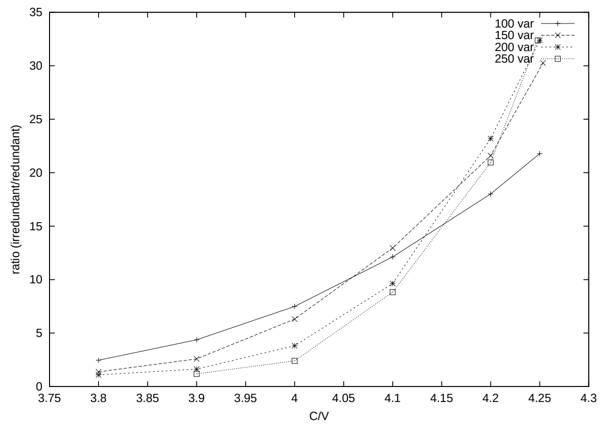

Figure 2: Mean ratio (irredundant/redundant) of the number of flips in walksat as a function of C/V and for different numbers of variables

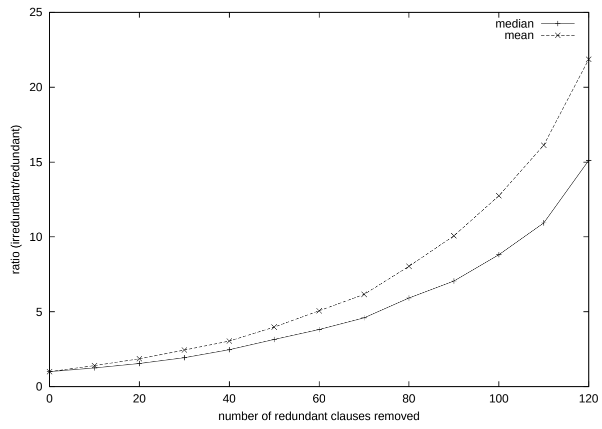

Figure 3: Median and mean ratio (irredundant/redundant) of the number of flips in walksat as a function of the number of redundant clauses removed (100 variables, 425 clauses)

to escape from these local minima. One can assert this rough principle: the more a formula has local minima the harder it is for walksat-like procedures. This is the case for irredundant formulas. Indeed, there exists, for every irredundant clause, a set of truth assignments that satisfies all the clauses of the formula except the latter one. Every such truth assignment is a good candidate for being a local minimum that is almost a solution. In an irredundant formula all the clauses have such a possible low local minima.

On satz procedure

We tested the performances of algorithms based on the DPL procedure (Davis & Putnam 1960; Davis, Logemann, & Loveland 1962) such as CSAT (Dubois *et al.* 1996; Boufkhad 1996), POSIT (Freeman 1995), NTAB (Crawford & Auton 1996). We report the results obtained with one of the most recent : satz (LI & Anbulagan 1997). The same conclusions derived here apply to the above algorithms. The figures 4 and 5 represent the mean and the median of the ratio of the number of branches needed by satz to solve a formula F_{irred} to the number of branches needed to solve F. Irredundant formulas prove to be harder also for satz, in a range of clauses to variables ratio that depends on the number of variables and that is in the vicinity of the phase transition. The same remark made about the relative positions of the curves in the case of walksat, apply to satz. Figure 6 shows that the difficulty of solving increases linearly as a function of the number of redundant clauses removed.

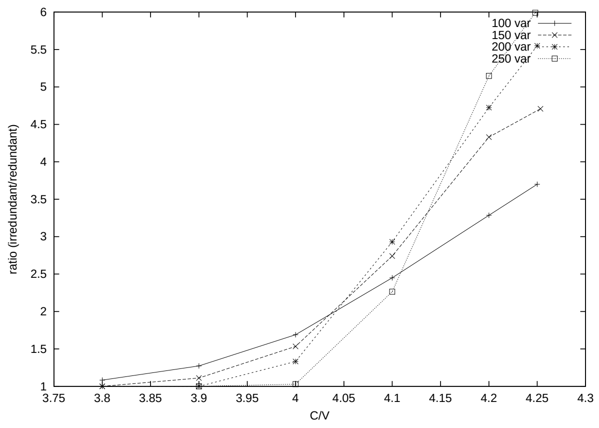

Figure 4: Median ratio (irredundant/redundant) of the number of branches in a satz tree as a function of C/V and for different numbers of variables

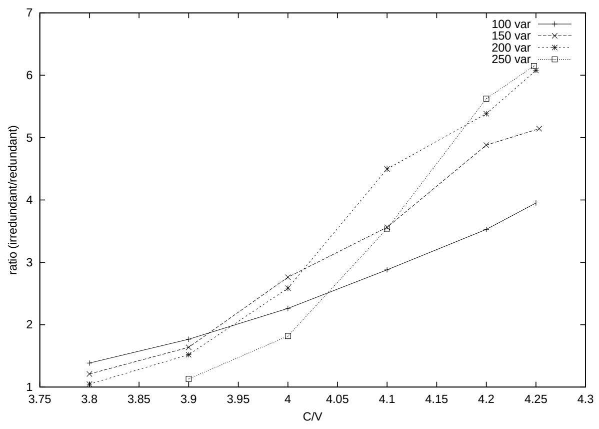

Figure 5: Mean ratio (irredundant/redundant) of the number of branches in a satz tree as a function of C/V and for different numbers of variables

We compared, in addition, the set of atoms of F and of F_{irred}. In a majority of formulas they were equal, and were almost equal in the few remaining formulas. This is important to understand the difference of hardness between F and F_{irred}. Indeed, let us denote by $V(F)$ the set of variables of a formula F. If $V(F) = V(F_{irred})$ then for every tree generated by a DPL-like procedure for F_{irred}, there exists an equal or shorter tree for F. This is true because the clauses that are in F but not if F_{irred} may cut some nodes in the tree

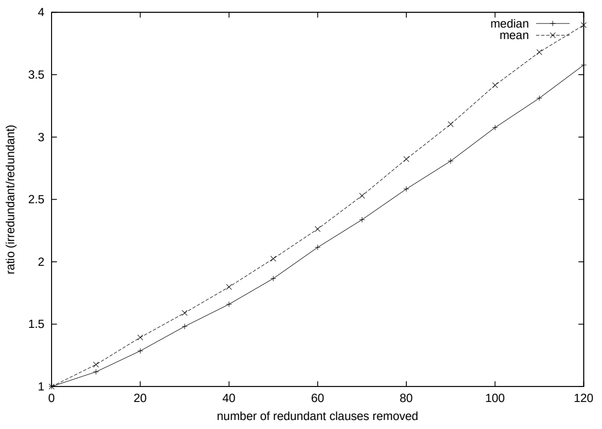

Figure 6: Median and mean ratio (irredundant/redundant) of the number of branches in a satz tree as a function of the number of redundant clauses removed (100 variables, 425 clauses)

of F_{irred}. In addition, if F_{irred} is satisfiable then any implicant of F_{irred} is an implicant of F and the possible node of the solution in the tree of F_{irred} need not to be extended to satisfy the clauses in $F - F_{irred}$.

When F is inconsistent, F_{irred} is an inconsistent kernel of F that is harder to solve than F, though there are methods that exploit the existence of an inconsistent kernel (Mazure, Saïs, & Grégoire 1996; Bayardo & Schrag 1996) to speed up proving the inconsistency of a formula. This leads us to give a necessary condition for an inconsistent kernel to be helpful for solving methods (which is not the case of F_{irred} with respect to F). Given an inconsistent formula F and an inconsistent subset F_K of clauses of F such that $V(F_K)$ is a proper subset of $V(F)$, F_K may be a helpful inconsistent kernel of F since in the tree of F, the nodes that involve only the variables of the set $V(F) - V(F_K)$ can be collapsed. As a conclusion a helpful inconsistent kernel of a formula F must discard variables from F to be possibly helpful.

Redundancy in random formulas

Now that we know that irredundant formulas are much harder, a question that may arise is: how does irredundancy vary in random 3SAT instances? To answer this question, we have taken, as measure of redundancy, for a formula F generated according to the 3SAT model, the ratio $\rho = 1 - \frac{|F_{irred}|}{|F|}$ called level of redundancy. F_{irred} is computed as described in the previous section. We recall that the chronological order used in the removal of redundant clauses doesn't modify the statistical distribution of ρ since formulas F have equal chances to be generated with any ordering of clauses.

As a function of the number of variables

The curve figure 7 represents the variations of the level of redundancy as a function of the number of variables, the ratio C/V being fixed and equal to 4.25 the approximate position of the phase transition for 3SAT. This variation shows clearly a decrease in the level of redundancy which tends to 0 when the number of variables tends to infinity. Together with the conclusions of the previous section, the fact that formulas tend to be irredundant with increasing values of the number of variables, shows clearly that, when the number of variables increases, the formulas tend to be harder not only because the number of variables increases but also because they become less redundant.

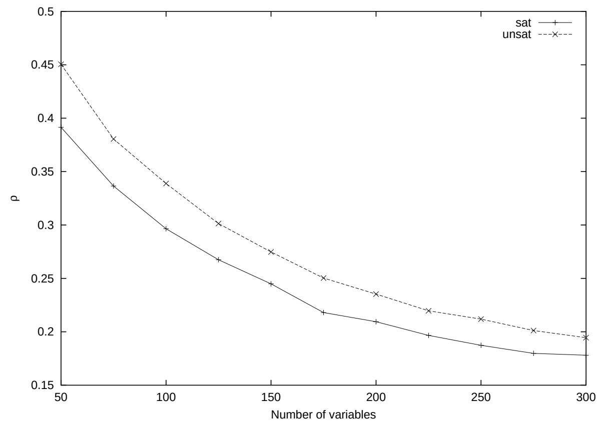

Figure 7: Median level of redundancy ρ for satisfiable and unsatisfiable formulas as a function of the number of variables and for C/V=4.25

For a formula F at the phase transition ratio 4.25, F_{irred} is located in the under-constrained region and can be considered as an exceptionally hard instance (EHI for short) (Gent & Walsh 1993; Hogg & Williams 1994). For 50 variables, an unsatisfiable formula in the phase transition has in average a level of redundancy of 0.46. In constrast, an equivalent irredundant formula would be located in average at a ratio equal to 2.3. For unsatisfiable formulas of 200 variables, the level of redundancy is 0.24. The irredundant equivalent formulas would be located in average at a ratio equal to 3.23 . This is a possible explanation for the fact that EHIs were surprisingly found in (Gent & Walsh 1993) at a ratio C/V between 1.8 and 3 for 50 variables while in (Crawford & Auton 1996) no EHI is found at the same range of ratios for 200 variables.

As a function of the clauses to variables ratio

Figure 8 represents the variations of ρ as a function of the C/V for different values of the number of variables. The level of redundancy is equal to 0 for small values of clauses to variables ratio then first starts to increase from a value of C/V that depends on n and which we note by $c_{red}(n)$. From there, the level of redundancy increases until the phase transition value. After that, the level of redundancy increases linearly as a function of the number of clauses since every clause added to almost every formula is redundant because almost every formula is then inconsistent.

The threshold of emergence of redundant clauses $c_{red}(n)$ increases in function of the number of variables and we con-

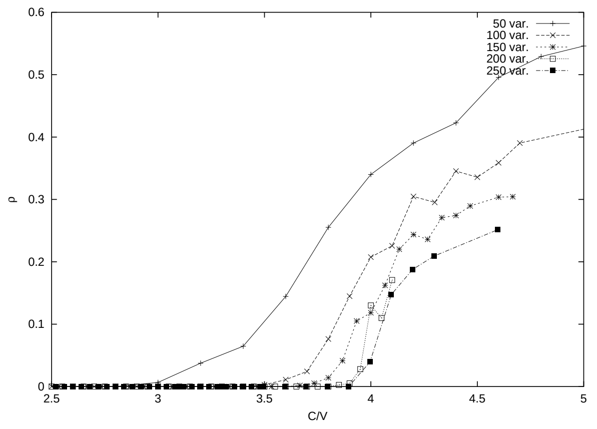

Figure 8: Median level of redundancy ρ for satisfiable and unsatisfiable formulas as a function of C/V and for different numbers of variables

jecture that it tends to the phase transition ratio. The existence of this threshold is to be related to the length of prime implicates and their number. Indeed, if the length of the shortest prime implicates that are not among the clauses of the formula F is greater than the length of the clauses of F then the probability that a newly generated clause will be redundant if added to F is equal to 0, since to be redundant if added to some formula F, a clause must be an implicate of F. Thus the threshold in the emergence of redundancy is certainly connected to the threshold of emergence of prime implicates as experimentally shown in (Schrag & Crawford 1996). In contrast, if F is inconsistent then the probability that a newly generated clause is redundant is equal to 1. Between the two extreme situations, this probability increases as the length of the shortest prime implicates decreases and their number increases.

When talking about the phase transition phenomenon in kSAT three regions are, generally, identified and referred to as under-constrained, critically constrained and over-constrained regions. The existence of these regions is related to the number of constraints. But since this number monotonically increases, it is not sufficient to explain the non monotonicity in the hardness. We give a possibly more accurate picture taking into account, in addition to the number of constraints, the level of redundancy. The formulas with few clauses are irredundant but have few constraints which make them easy. The formulas with a lot of clauses are very constrained but are highly redundant which makes them easy. Between these two situations we have constrained and nearly irredundant formulas which are the hardest. To sum up, the formulas at the phase transition can be considered as located in the cross-over between a decreasing irredundancy and an increasing number of constraints.

An irredundant formulas generator

We describe in this section an algorithm for generating random irredundant formulas. The set of clause is initialized to the empty set then is built by iteratively adding randomly generated clauses after checking at each step that the resulting formula is irredundant. The main problem that has to be faced is that we have to test, at each step, not only the redundancy of the newly generated clause but also if adding this clause does not make some other clause, already in the formula, redundant. To avoid doing systematically redundancy tests for every clause, we use the following fact: for every irredundant clause C in a formula F there exists an implicant I of $F - \{C\}$ that contradicts C. We call such an implicant a *witness* of the redundancy of C in F. Let us suppose that, at some point in the generating process, the current formula F is irredundant and we have a witness for every clause. When a new clause C is generated, its redundancy is tested. If it is irredundant then the algorithm tests if it is satisfied by every witness of the clauses already generated. If it is not satisfied by some witness, we must check if the corresponding clause is redundant and, if it isn't, we must find another witness which satisfies C. The algorithm first tries to modify the witness, by adding to it new literals, if possible, to satisfy C. If this is not possible then a completely new witness is searched. If none exists the algorithm rejects the clause C. These steps are detailed below:

- Check the satisfiability of $F \cup \{\overline{C}\}$, if it is unsatisfiable then reject C since it is redundant, otherwise record the implicant of $F \cup \{\overline{C}\}$, found by the previous satisfiability test, as a witness for C.
- If C is not redundant, check if it is satisfied by every witness of the clauses of F. For every witness w_i that does not satisfy C (w_i corresponding to some clause C_i):
  1. either there exists a literal of C such that its underlying variable is not in w_i, in which case add this literal to w_i so that w_i satisfies $F \cup \{C\} - \{C_i\}$.
  2. There exists no such literal then check the redundancy of C_i in $F \cup \{C\}$. If it is irredundant then the implicant found by this test will replace the witness of C_i otherwise reject C and reset the witnesses modified by this step to their previous values.

At the end of these steps the clause C, if not rejected, is added to the current set of clauses. This algorithm is not guaranteed to terminate because it may happen that at some step no clause maintaining the irredundancy is, after many attempts, randomly selected. One can limit the number of attempts for finding a clause that maintains the irredundancy if added. If this limit is reached, the generator answers that it has failed to generate an irredundant formula at the given number of clauses. This will, for example, stop the algorithm if the formula is unsatisfiable before the required number of clauses is reached.

Conclusion

We have empirically shown that redundancy is a characteristic that conditions the hardness of the random formulas. We have given results which show that irredundant formulas are harder than redundant ones both for local search procedures and proof procedures such as DPL-like procedures. We also have exhibited that random formulas become less and less redundant as their size increases. Since these formulas are used as benchmarks to compare algorithms and to choose the best settings for their parameters, one has to be careful no to exploit this characteristic to improve an algorithm. If an algorithm A exploits only the redundancy to improve over an algorithm B then the performances of A and B will converge when the number of variables increases. We suggest to compare algorithms on irredundant random formulas and to try to improve algorithms on these formulas, instead. To this end we have given an algorithm of a generator of irredundant formulas which avoids some clause redundancy checks.

This work can be generalized by studying the redondancy in realistic problems and its impact on their hardness. It would also be interesting to identify the possible other characteristics of this type so that the challenging small size formulas for SAT algorithms will be structurally identical to the larger ones.

Acknowledgments

This work has been supported in part by the I.U.T. de Lens and the “Region Nord/Pas-de-Calais”.

References

Bayardo, R. J., and Schrag, R. 1996. Using CSP look-back techniques to solve exceptionally hard SAT instances. In Springer., ed., *Proc. of the Second Int’l Conf. on Principles and Practice of Constraint Programming*, 46–60.

Boufkhad, Y. 1996. *Aspects Probabilistes et Algorithmiques du Problème de Satisfiabilité*. Ph.D. Dissertation, Université Paris 6.

Crawford, J. M., and Auton, L. 1996. Experimental Results on the Crossover Point in Random 3-SAT. *Artificial Intelligence* 81.

Davis, M., and Putnam, H. 1960. A computing Procedure for Quantification Theory. *Jour. Assoc. for Comput. Mach.* 7:201–215.

Davis, M.; Logemann, G.; and Loveland, D. 1962. A Machine Program for Theorem Proving. *Jour. Assoc. for Comput. Mach.* 5:267–270.

Dubois, O.; André, P.; Boufkhad, Y.; and Carlier, J. 1996. SAT versus UNSAT. In *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, number 26. American Mathematical Society. 415–436.

Freeman, J. W. 1995. *Improvements to Propositional Satisfiability Search Algorithms*. Ph.D. Dissertation, University of Pennsylvania, Philadelphia.

Gent, I. P., and Walsh, T. 1993. Easy Problems are Sometimes Hard. *Artificial Intelligence* 70:335–345.

Hogg, T., and Williams, C. 1994. The Hardest Constraint Problems: A Double Phase Transition. *Artificial Intelligence* 69:359–377.

Larabee, T., and Tsuji, Y. 1993. Evidence for a Satisfiability Threshold for Random 3CNF Formulas. In *Proceedings AAAI Spring Symposium*, 112–118.

LI, C. M., and Anbulagan. 1997. Heuristics Based on Unit Propagation for Satisfiability Problems. In *Proceedings of IJCAI’97*, 366–371.

Mazure, B.; Saïs, L.; and Grégoire, E. 1996. Detecting Logical Inconsistencies. In *Annals of Mathematics and Artificial Intelligence*.

McAllester, D.; Selman, B.; and Kautz, H. 1997. Evidence for Invariants in Local Search. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence AAAI’97*.

Mitchell, D.; Selman, B.; and Levesque, H. 1992. Hard and Easy Distribution of SAT Problems. In *Proceedings of AAAI’92*, 459–465.

Schrag, R., and Crawford, J. M. 1996. Implicates and Prime Implicates in Random 3SAT. *Artificial Intelligence* 81:199–222.