Let the Solver Deal with Redundancy

Cédric Piette

Let the Solver Deal with Redundancy

Cédric Piette
Université Lille-Nord de France, Artois
CRIL-CNRS UMR 8188
F-62307 Lens Cedex, France
piette@cril.fr

Abstract

Handling redundancy in propositional reasoning and search is an active path of theoretical research. For instance, the complexity of some redundancy-related problems for CNF formulae and for their 2-SAT and Horn SAT fragments have been recently studied. However, this issue is not actually addressed in practice in modern SAT solvers, and is most of the time just ignored. Dealing with redundancy in CNF formulae while preserving the performance of SAT solvers is clearly an important challenge. In this paper, a self-adaptative process is proposed to manage redundant clauses, enabling redundant information to be discriminated and to keep only the one that proves useful during the search.

1 Introduction

SAT is the NP-complete decision problem that consists in checking whether a set of propositional clauses (also called CNF formula) admits at least one truth assignment that satisfies all clauses. This problem is of central importance in computer science and has many practical applications in various areas, such as electronic design automation, artificial intelligence, bioinformatics or formal verification to name of few. There is a large and very active research community involved with both the theoretical and experimental algorithmic studies of SAT (see e.g. `http://www.SATlive.org`), and its applications.

Problems related to redundancy within CNF formulae have been well-studied in the literature, especially from a theoretical point of view. Indeed, the problem of minimizing redundant subformulae in a propositional formula was already analyzed within the context of the first formalization of the polynomial hierarchy [17]. Complexity results about redundancy-related computational problems have been established in [13]. Other results have also been obtained for restricted cases, for instance when the formula is composed of Horn clauses, exclusively (see e.g. [1, 11]). Later, the importance of redundancy for practical SAT solving has been emphasized by several studies. A first one, conducted on random 3-SAT formulae, provides empirical results about the role of redundancy in CNF formulae. Especially, it is claimed that irredundant CNF formulae are typically harder to solve than the same formulae augmented with redundant information. This work has been extended later [20], particularly showing that the hardness of CNF formulae solving is related to the size of their irredundant subformulae.

Despite those studies that show strong relations between redundancy and practical solvability, redundancy is an issue that is ignored in practice in current solvers, since it would first require redundant clauses to be extracted. Unfortunately, this extraction is an heavy task in the worst case. Indeed, redundancy-related problems are at least in the first level of the polynomial hierarchy, which makes them as hard as SAT itself. Checking whether a given clause can be inferred from the remaining part of the formula is CoNP-complete. In this paper, a practical approach to deal with clause redundancy during the search for satisfiability is investigated and experimentally evaluated. First, to circumvent the high complexity of redundancy checking, an incomplete but linear-time inference process is used. Secondly, as eliminating or adding redundant clauses can either improve or degrade the performance of the solver, each clause of a the CNF formula is checked for redundancy and added to the learnt database while leaving the modern SAT solver the freedom to use or eliminate them according to their activities during search.

The paper is organized as follows. In the next Section, basic clausal propositional logic and redundancy-related concepts are provided. A new scheme for SAT solving for dealing with redundancy is introduced in Section 3. In Section 4, an empirical study proves the usefulness of the approach. Finally, we conclude with some promising paths for future research.

2 About redundancy in CNF formulae

Let $\mathcal{L}$ be a standard Boolean (also called propositional) logical language built on a finite set of Boolean variables. A *CNF formula* is a set (interpreted as a conjunction) of clauses, where a *clause* is a set (interpreted as a disjunction) of literals. A *literal* is a propositional variable x (which can be either $true$ or $false$) or its negation $\neg x$. The two literals x and $\neg x$ are called *complementary*. We note $\bar{l}$ the complementary literal of l. Let L be a set of literals, $\bar{L}$ is defined as $\{\bar{l} \mid l \in L\}$. A *unit* clause is a clause containing one literal (called *unit literal*), only. A binary clause contains exactly two literals. An *empty clause*, noted $\bot$, is interpreted as $false$ (unsatisfiable), whereas an *empty CNF formula*, noted $\top$, is interpreted as $true$ (satisfiable).

The set of variables occurring in Σ is noted V_Σ. A set of literals is *complete* if it contains one literal for each variable in V_Σ, and *fundamental* if it does not contain complementary literals. An *interpretation* ρ of a propositional formula Σ associates a value $\rho(x)$ to the variables $x \in V_\Sigma$. An interpretation is also represented by the complete and fundamental set of literals that it satisfies. A *model* ρ of a formula Σ is an interpretation that satisfies Σ, noted $\rho \models \Sigma$.

Let Σ be a CNF formula. We note $\Sigma|_x$ the formula obtained from Σ by assigning the literal x the truth-value $true$. Formally $\Sigma|_x = \{c \mid c \in \Sigma \text{ and } \{x, \neg x\} \cap c = \emptyset\} \cup \{c \backslash \{\neg x\} \mid c \in \Sigma \text{ and } \neg x \in c\}$. This notation is extended to interpretations: given an interpretation $\rho = \{x_1, \ldots, x_n\}$, $\Sigma|_\rho$ is defined as $(\ldots((\Sigma|_{x_1})|_{x_2})\ldots|_{x_n})$. We also note Σ^* the formula Σ closed under unit propagation (UP), defined recursively as follows:

1. $\Sigma^* = \Sigma$ if Σ does not contain any unit clause,
2. $\bot \in \Sigma^*$ if Σ contains two unit-clauses $\{x\}$ and $\{\neg x\}$,
3. otherwise, $\Sigma^* = (\Sigma|_x)^*$ where x is the literal appearing in a unit clause of Σ.

Moreover, a clause c is implied by unit propagation from Σ, noted $\Sigma \models^* c$, if $\bot \in (\Sigma|_{\bar{c}})^*$.

The way a problem is encoded in CNF is now recognized as crucial for its practical resolution, and many studies have been devoted to this point (see e.g. [12]). Particularly, the presence of redundant information can play an important role for determining its satisfiability. Indeed, expliciting a lot of implied constraints can actually help solvers. Nevertheless, recording redundant information finds its limits in memory space constraints. As a matter of fact, at any step of a search for satisfiability by most current DPLL-based solvers [4], two unassigned literals of each clause are *watched* and an incomplete list of occurrences has to be updated after each assignment, following the principles of lazy data structures [21]. Accordingly, the presence of a large number of redundant clauses just slows down the resolution process by providing too much information to manage.

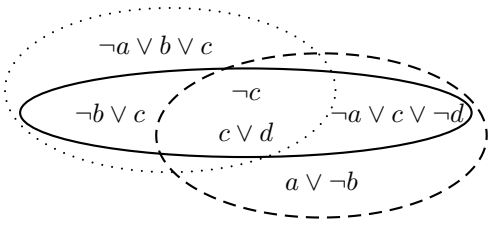

Figure 1. The 3 irredundant subformulae of Example 2

Let us define a redundant clause formally.

Definition 1 *Let Σ be a CNF formula and c a clause such that $c \in \Sigma$. c is called* redundant *(w.r.t. Σ) if and only if $\Sigma \backslash \{c\} \models \{c\}$. Any non redundant clause is called* irredundant.

Through this definition, it is clear that when a clause is redundant, it can be safely removed from the formula while preserving satisfiability.

Example 1 *Let $\Sigma = \{\neg a \vee b \vee c, \neg b \vee c, \neg c, a \vee \neg b, c \vee d, \neg a \vee c \vee \neg d\}$ be a CNF formula. The clause $\phi_1 = \neg b \vee c$ is redundant w.r.t. Σ, since $(\Sigma \backslash \{\phi_1\})|_{\bar{\phi_1}}$ is clearly unsatisfiable. The clauses ϕ_2=$\neg a \vee b \vee c$, ϕ_3=$a \vee \neg b$ and ϕ_4=$\neg a \vee c \vee \neg d$ are also redundant w.r.t. Σ.*

Checking whether a clause is redundant or not is clearly CoNP-complete. By iterating a redundancy test on each clause of a CNF formula and removing it when it has been proved redundant, one obtains an *irredundant formula*.

Definition 2 *Let Σ be a CNF formula. Σ is called* irredundant formula *if and only if $\forall c \in \Sigma$, c is irredundant.*

This notion of irredundant formula has been used for a long time, even if it has been named in different ways in the literature. For example, it is called *irredundant equivalent subset* in [13] and *satisfiable core* in [5]. Moreover, many studies have recently focused on the problem of explaining *why* a CNF formula does not exhibit any solution, through the concept of Minimally Unsatisfiable Subformula, or MUS (see e.g. [10, 22, 9]). An MUS is actually an irredundant formula for the special case of unsatisfiability.

A CNF formula can clearly exhibit several irredundant subformulae; actually, a CNF formula with m clauses exhibits $C_m^{m/2}$ such irredundant subformulae, in the worst case.

Example 2 *Let Σ be the CNF formula from the previous example. Σ exhibits 3 irredundant sub-formulae, which*

are $\Sigma_1 = \Sigma\backslash\{\phi_1, \phi_2\}$, $\Sigma_2 = \Sigma\backslash\{\phi_2, \phi_3\}$ *and* $\Sigma_3 = \Sigma\backslash\{\phi_3, \phi_4\}$. *Those 3 formulae are illustrated in Figure 1.*

In the general case, the role of redundant clauses within CNF formulae remains unclear w.r.t. SAT solving. In [5], it is claimed that irredundant formulae are in general harder to solve than the same formulae augmented with redundant clauses. Indeed, it is well-known that redundant information can actually help SAT solvers; for instance, the powerful learning scheme introduced in [3], which produces a particular resolvent clause after each conflict, can be viewed as a dynamic addition of redundant clauses during the search process. This learning strategy is now recognized to be one of the key features of modern SAT solvers, which proves the interest of redundant information with respect to practical SAT solving. However, it is also well-known that adding too many redundant clauses can make the search computationally heavier by producing too much information to manage. Besides, a simple experiment that consists in adding all learnt clauses to a CNF formula after its resolution shows that this new redundant information makes the formula generally more difficult to solve.

Hence, in the general case it appears difficult to predict whether a redundant clause will prove useful for practical resolution of a CNF formula, particularly because it depends on the way the search space is explored. In the next Section, an approach to deal with redundant information is proposed.

3 A new scheme for managing redundancy

3.1 Extracting redundant clauses polynomially

As shown in the last section, redundant information can prove very useful for practical SAT solving. However, it can also clutter the resolution process. We propose here an original way to deal with this redundant information. Unfortunately, this requires us to be able to extract redundant clauses within CNF formulae, and checking whether a clause is redundant is a CoNP-complete task; performing this test as such does thus not make sense when trying to increase the efficiency of SAT solvers. Yet, a recent approach [7] aims at detecting – in an incomplete way – redundant clauses in polytime. The main idea is to perform the redundancy test through unit propagation, only.

Definition 3 *Let* Σ *be a CNF formula and* $c \in \Sigma$ *be a clause.* c *is called* U-redundant *if and only if* $\Sigma\backslash\{c\} \models^* c$ *i.e.* $\bot \in (\Sigma\backslash\{c\})|_{\bar{c}})^*$.

Considering U-redundant clauses enables to avoid any combinatorial explosion while detecting a great number of redundant clauses in practice. Actually, any formula Σ can be decomposed in a polynomial time into $\Sigma_{uir} \cup \Sigma_{ur}$, where

Input: Σ : a CNF formula
Output: $true$ if Σ is satisfiable, $false$ otherwise

```
1  begin
2      Σ_ur ← ∅ ;
3      Σ_uir ← Σ ;
4      foreach c ∈ Σ do
5          Σ_uir ← Σ_uir \ {c} ;
6          Let c = {l_1, ..., l_i, ..., l_k} ;
7          if (Σ_uir|_{l̄_1,...,l̄_i})* = ⊥ then
8              Σ_ur ← Σ_ur ∪ {l_1, ..., l_i} ;
9          end
10         else
11             Σ_uir ← Σ_uir ∪ {c} ;
12         end
13     end
14     return solve(Σ_uir,Σ_ur) ;
15 end
```

Algorithm 1: U-redSAT solver

Σ_{uir} is a U-irredundant sub-formula of Σ and Σ_{ur} is a set of redundant clauses w.r.t. Σ. Obviously, different clause orderings for redundancy checking may lead to different U-irredundant sub-formulae.

3.2 Special usage of detected redundant clauses

Our intuition is that a well-known feature of modern SAT solvers could be in charge of selecting relevant redundant clauses with respect to the state of the search in progress. Indeed, current solvers produce redundant information after each conflict through their learning functions. The problems of storing and updating clauses have then already been addressed: different well-tuned techniques ensure a good trade-off between the amount of information to store/update and the propagation capacity of the algorithm. First, it was proposed to only keep the clauses whose size is less than a given value, provided that the shorter a clause is, the more unit propagations it should trigger [16]. However, the most widely used technique in current solvers is inspired by the branching heuristic VSIDS [18], and aims at keeping the most active clauses, i.e. clauses that contain literals involved in the greatest number of conflicts [8]. More precisely, a counter (initialized to 0) is associated with each learnt clause, and is incremented when the corresponding clause has played a role with respect to an encountered conflict. Periodically, the learnt clauses base is purged from the ones that exhibit the lowest values, following the assumption that the most useful clauses until now will also be the most useful for the rest of the search.

Hence, we propose to use the method of Fourdrinoy *et al.* [7] to efficiently capture redundant clauses within CNF formulae, and inform the solver of the nature of those detected

clauses. To this end, the detected redundant clauses are eliminated from the CNF formula and moved to the learnt database of the SAT solver.

Thanks to this approach, the most useful (with the highest activity) redundant clauses will be kept, whereas the non relevant ones will be progressively deleted. In this way, redundant clauses not relevant for the current proof are dynamically eliminated, leading to an interesting improvement in both space and time complexity.

Our proposed approach is depicted in Algorithm 1. First, all the clauses of Σ are checked for redundancy (line 4 to 13). Let us note that in practice, the test in line 7 is performed by alternating the assignment of the opposite of one literal from c (literal l_i with $1 \leq i \leq k$) with unit propagation. Hence, if a conflict occurs without all opposites having been assigned ($i < k$), the clause made of the first i literals of c is deduced and added to Σ_{ur}.

At the end of this preprocessing step, two formulae Σ_{uir} and Σ_{ur} are obtained. Let us now assume that a SAT solver `solve` is modified in order to record its first parameter as the input formula and the second one as an "initial" learnt clauses database: a classical call to such a solver would be "`solve`(Σ,$\emptyset$)". Instead of such a call, after this preliminary step the modified SAT solver is called using the two formulae Σ_{uir} and Σ_{ur} (line 14). Let us note that those CNF formulae obey the following property.

Property 1 *Let Σ be a CNF formula, and $\Sigma_{uir}, \Sigma_{ur}$ the partition of Σ delivered by the first part of Algorithm 1. We have:*

1. *Σ and Σ_{uir} are equivalent with respect to satisfiability*
2. *$\Sigma_{ur} \vDash \Sigma \backslash \Sigma_{uir}$*

Proof

1. *Straightforward: by definition, eliminating a redundant clause preserves satisfiability. A process that iterates a test of redundancy and deletes each clause of a CNF formula when it is redundant clearly returns an equivalent formula with respect to satisfiability*
2. *Let us assume that a clause $c = \{l_1, \ldots, l_i, \ldots, l_k\} \in \Sigma$ is U-redundant. As the literals of $\bar{c}$ are assigned one by one, and unit propagation is applied at each assignment, one can reach the conflict after the assignment of the literal l_i i.e. $\bot \in (\Sigma|_{\{\bar{l_1}, \ldots, \bar{l_i}\}})^*$. In such a case, we have $\Sigma \vDash^* \{l_1, \ldots, l_i\}$. Σ_{ur} is then composed of subclauses of $\Sigma \backslash \Sigma_{uir}$, hence this latter set of clauses can clearly be infered by the former one.*

This schema of SAT solving, from redundancy checking to the special storage of U-redundant clauses, can be easily grafted to most of current modern SAT solvers. In Section 4, these ideas are evaluated, from an experimental point of view.

4 Experimental results

In order to assess the pertinence of those ideas, we have implemented the unit-redundancy checking procedure to compute (in an incomplete way) a set of redundant clauses within a CNF. As the order of the tested clauses matters, we have chosen to sort the clauses w.r.t. their decreasing size, as suggested by [7], to obtain a formula closed under subsumption. As a result, the procedure delivers a partition of two sets of clauses, the first part containing the fundamental clauses whereas the second one is a set of redundant clauses.

Then, we have modified a complete method in order to take this information into account by considering the detected redundant clauses as learnt ones. We have compared the behavior of this modified solver with the original version, using all clauses of the formula as fundamental (no one could be deleted, and all of them are all considered during the whole computation). As a case study, `minisat` [6] was selected as the complete solver. As a point of technical clarification, let us note that the parameters of `minisat` were voluntarily kept as such in our modified version, in order to obtain a fair comparison between solvers. Especially, the number of recorded clauses during the search was initially set to $|\Sigma|/3$. This means that at any time, both solvers exhibit the same upper bound on the number of learnt clauses at any time, which contributes to run them under the same conditions. Nevertheless, our version starts the computation with a non-empty set of learnt clauses, and when the cardinality of this set exceeds a third of the number of clauses, some of them are removed as soon as a conflict is reached. Fortunately, as our experimental results show, this case occurs only a very few times.

Our experiments have been conducted on a selection of 940 real-world (crafted and industrial) problems from the last SAT competitions [19]. First, a sample of the results is proposed in Table 1, which is composed of 3 main columns. The first one provides information about the tested instances, through their names together with their numbers of clauses and their satisfiability status (S for SAT, U for UNSAT). The second column of the table focuses on the redundancy test, reporting the number of redundant clauses detected by the approach, the percentage of clauses it represents and the needed time in second. Finally, in the third column, the time (in seconds) needed to solve the formula, using either the original `minisat` or the modified version ($\texttt{minisat}_{red}$) taking into account the nature of the redundant clauses is reported. For $\texttt{minisat}_{red}$, preprocessing *and* solving times are summed and reported. Our experimental studies have been conducted on Intel Xeon 3GHz under Linux CentOS 4.1. (kernel 2.6.9) with 2GB of RAM. For all these experimentations, a time-out limit of 900 seconds was respected. If a computation exceeded this limit,

Instance			Redundancy test			Solving time (sec.)	
Name	#cla	S/U	#cla$_{red}$	$\%_{red}$	time (sec.)	minisat	minisat$_{red}$
hanoi5u	59,718	U	2,139	3.58%	1.21	214.87	164.15
bqwh.33.381	9,040	S	494	5.46%	0.06	456.5	38.32
5col100_15_6	3,473	U	397	11.43%	0.03	56.93	30.88
lksat-n2000-m6840-k3-l4	6,598	U	242	3.66%	0.02	28.72	29.66
Composite-024BitPrimes-0	9,689	S	793	8.18%	0.05	206.25	163.71
shuffling-1-s1025511904	42,818	U	4,818	11.3%	2.17	284.04	332.98
manol-pipe-c6id	238,142	U	3,899	1.64%	3.38	150.58	147.97
ferry12	23,285	S	7,285	31.3%	0.43	0.85	0.94
avg-checker-5-34	33,206	U	2,700	8.13%	1.09	55.09	28.77
2dlx_cc_mc_ex_bp_f2_bug015	149,693	S	37,664	25.2%	40.35	2.31	2.35
9vliw_bp_mc	156,921	U	22,542	14.4%	59.99	248.77	898.7
k2fix_gr_2pinvar_w8	270,136	U	0	0%	30.04	67.52	67.48
hanoi6	22,633	S	11,120	49.1%	0.21	132.56	87.69
shuffling-2-s1182968979	58,408	U	4,487	7.68%	5.05	405.65	348.83
frg2mul.miter	58,477	U	4,418	7.56%	2.02	540.28	305.04
CompositeRSA640	1,929,086	?	105,148	5.45%	26.04	*time out*	*time out*
lksat-n900-m6174-k4-l4-s...	6,084	U	90	1.48%	0.02	57.59	109.26
4pipe_1_ooo	60,564	U	13,956	23.04%	4.19	184.56	82.56
rand_net70-30-5	12,071	U	390	3.23%	0.33	266.33	344.21
gripper10	14,126	S	3,782	26.77%	0.21	*time out*	192.95

Table 1. Redundancy rate and solving time on a sample of benchmarks

then a *time out* was reported.

First, let us focus on the unit-redundancy test. The obtained results show that there exists modelizations of problems that generate a really large number of redundant clauses. For example, the `2dlx...bug015` and `ferry12` instances exhibit at least 25.2% and 31.3% of their clauses that are redundant, respectively. Limiting this test to unit propagation thus enables to detect a lot of such clauses and leads very often to a cheap computation: most of the time, this test can be performed in less than 5 seconds, even if, for huge instances, several tens of seconds can be necessary (`CompositeRSA640`). The most redundant CNF formula discovered is the clausal propositional encoding of the Hanoi tower problem of size 6.This benchmark contains 22633 clauses, but about half of them are implied by the remaining part of the CNF formula and are not actually necessary to encode the problem. Fortunately, for most tested benchmarks, this "redundancy rate" did not exceed 15%. Actually, in some cases, the procedure was not able to detect any redundant clause at all (e.g. `k2fix_gr_2pinvar_w8`).

Now, let us consider the consequences of using this polynomially obtained information. Discriminating redundant clauses in order to eventually lighten the amount of data the solver has to update after each assignment (either by decision or propagation) speeds up the resolution of instances, very often. It has been observed that redundant clauses are sometimes kept by the solver a long time during the search, or are erased very quickly, when more helpful clauses are learnt by the solver. Thus, the adaptative storage of clauses of modern solvers also appears to be appropriate to manage clauses known to be non fundamental. Note that even when the CNF formula exhibits a quite small redundancy rate ($< 15\%$), using those clauses as learnt ones often boosts the solver (`Composite-024BitPrimes-0`, `avg-checker-5-34`). Taking the redundancy of clauses into account can sometimes be less efficient that just ignoring their redundancy status. But on many cases, computing this strategic information proves useful for practical SAT solving. Obviously, on some cases, taking the redundancy of clauses into account is less efficient than just ignoring their redundancy status. For instance, in spite of the discovery of the redundancy of 14.3% of its clauses, the original `minisat` proves the unsatisfiability of `9vliw_bp_mc` in about 4 minutes whereas making those clauses optional for the same solver leads to solve of formula in almost 15 minutes. Nevertheless, on many cases, computing this strategic information proves useful for practical SAT solving.

In a more general way, we have compared the needed solving time of our modified version of `minisat` with the original one. The obtained results are provided in Figure 2. Each plot in this figure represents the time for solving a particular CNF formula: the time for `minisat`$_{red}$ is given by its projection on the X-axis, whereas the time for the original `minisat` is given by its projection on the Y-axis. Thus, a plot located above (below) the first diagonal means

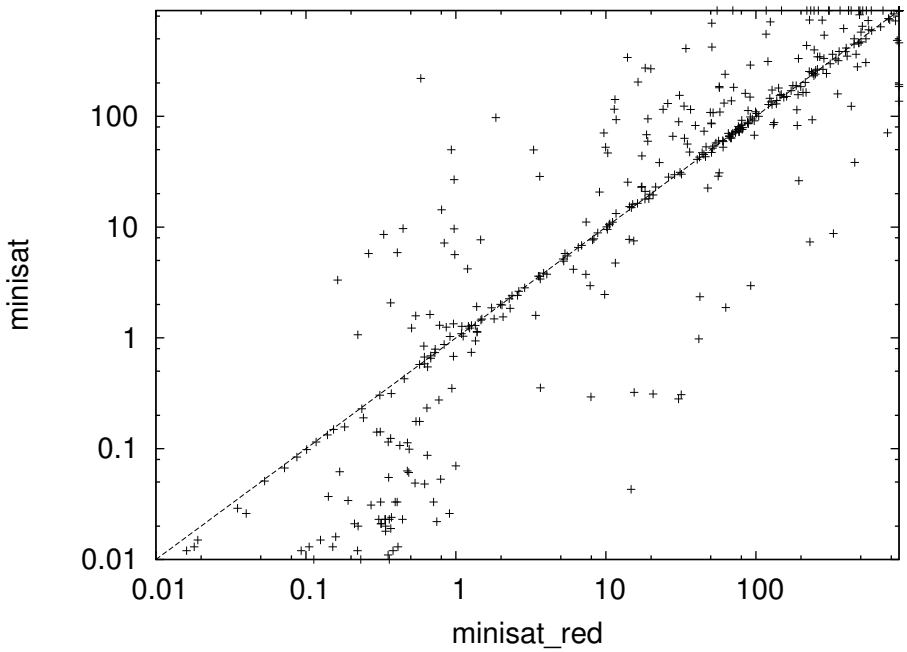

Figure 2. Minisat taking detected redundant clauses into account (or not)

that the modified version needed less (resp. more) time to solve the CNF formula than the original version. Let us also note that both axes are displayed in a logarithmic scale in Figure 2.

First, it is noticeable that for easy-to-solve problems (solving time less than 1 second), the "classic" `minisat` is generally the most efficient version. This phenomena is explained by the computational overhead due to the preliminary redundancy test. This one is most often computationally inexpensive, but for such easy problems, any amount of spent time matters. Moreover, as previously mentioned, for very large CNF formulae, the preliminary tests may require several seconds. When such a benchmark is really easy to be solved for CDCL implementations, useless computation of redundant clauses can deteriorate the results. Fortunately, this waste of time represents only a few tens of seconds in the worst empirically observed cases.

Let us now focus on more difficult benchmarks (solving time larger than 10 seconds). On such problems, selecting the detected redundant clauses w.r.t. their number of performed propagations proves valuable very often. Indeed, most of the plots are located above the diagonal, which shows the improvement of `minisat` thanks to our new feature. Using the VSIDS-like strategy on redundant clauses thus appears adapted for dealing with those particular extracted clauses.

Finally, let us also note that a lot of plots are located around the diagonal. This is explained by the fact that there exists some CNF formulae that exhibit only a few or even no U-redundant clauses. Obviously enough, in such cases the behavior of both versions of `minisat` is very similar, and is exactly the same when no U-redundant clauses are extracted by the preliminary detection procedure.

Nevertheless, those first results plead for more attention about redundancy of information within CNF formulae for practical SAT solving. Our first implementation clearly delivers promising results, and using existing mechanisms created for learnt clauses appears adapted for dealing with redundant clauses. Clearly enough, even better results could be expected by fine-tuning the various parameters of the solver, especially the initial number of stored learnt clauses.

5 Conclusions

In this paper, a new strategy to deal with redundant clauses within CNF formulae has been presented in the context of SAT solving. This technique has the great advantage to be easily graftable to current solvers, thanks to an original use of some of their features. More precisely, the idea is to inexpensively extract a set of redundant clauses within a CNF formula, and test whether each clause of the CNF formula is redundant w.r.t. the remaining part of the problem; accordingly, a set of redundant clauses is delivered and added to the learnt database. Therefore, the VSIDS-like strategy of state-of-the-art solvers is applied to them, and if they do not enable to propagate literals during the search, then the solver can just remove them. This technique can be viewed as an automatic tuning strategy to manage redun-

dancy within CNF formulae. It has been empirically validated through intensive experiments that show its practical interest.

This work opens many interesting future directions of research. First, as mentioned earlier in this paper, the order according to which the clauses are tested for redundancy has a great importance for obtaining an easy-to-solve CNF formula. The subsumption-based choice made here proves relevant, but new ones should be investigated. Moreover, SAT solvers often exhibit a large number of parameters that are crucial for the efficiency of the procedure. It has been chosen not to modify those parameters in the used proof-of-concept solver, but some of them, especially the number of allowed learnt clauses, could be redefined taking the initially provided extra information into consideration. Finally, this study focuses on the way redundant constraints existing within CNF formulae can be managed. It would be also interesting to produce redundant clauses by performing limited resolution, as preprocessors like `HyPre` [2] do, for instance. The produced clauses could be added within the solver as learnt, in order to help the solver to make propagations when they are actually useful. If some of them do not prove useful during the search, the solver would then be able to get rid of those clauses. We plan to explore those different paths of research in the next future.

Acknowledgement

The author would like to thank Bertrand Mazure and Lakhdar Saïs for very helpful discussions.

References

[1] G. Ausiello, A. D'Atri, and D. Saccá. Minimal representation of directed hypergraphs. *SIAM Journal on Computing*, 15(2):418–431, 1986.

[2] F. Bacchus and J. Winter. Effective preprocessing with hyper-resolution and equality reduction. In *6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919 of *Lecture Notes in Computer Science*, pages 341–355, Santa Margherita Ligure (Italy), 2003. Springer.

[3] P. Beame, H. Kautz, and A. Sabharwal. Understanding the power of clause learning. In *18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 1194–1201, 2003.

[4] L. Bordeaux, Y. Hamadi, and L. Zhang. Propositional satisfiability and constraint programming: A comparative survey. *ACM Computing Surveys*, 38(4), 2006.

[5] Y. Boufkhad and O. Roussel. Redundancy in random SAT formulas. In *17th National Conference on Artificial Intelligence (AAAI'00)*, pages 273–278, 2000.

[6] N. Eén and N. Sorensson. Minisat home page http://www.cs.chalmers.se/cs/research/formalmethods/minisat.

[7] O. Fourdrinoy, É. Grégoire, B. Mazure, and L. Saïs. Eliminating redundant clauses in SAT instances. In *4th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'07)*, volume 4510 of *Lecture Notes in Computer Science*, pages 71–83, Brussels (Belgium), 2007. Springer.

[8] E. P. Goldberg and Y. Novikov. Berkmin: a fast and robust SAT-solver. In *Design Automation and Test in Europe (DATE'02)*, pages 142–149, Paris (France), 2002.

[9] É. Grégoire, B. Mazure, and C. Piette. Boosting a complete technique to find MSS and MUS thanks to a local search oracle. In *20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, volume 2, pages 2300–2305, Hyderabad (India), 2007.

[10] É. Grégoire, B. Mazure, and C. Piette. Local-search extraction of MUSes. *Constraints Journal*, 12(3):325–344, 2007.

[11] P. L. Hammer and A. Kogan. Optimal compression of propositional horn knowledge bases: Complexity and approximation. *Artificial Intelligence*, 64(1):131–145, 1993.

[12] A. Hertel, P. Hertel, and A. Urquhart. Formalizing dangerous SAT encodings. In *10th International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 of *Lecture Notes in Computer Science*, pages 159–172, Lisbon (Portugal), 2007. Springer.

[13] P. Liberatore. The complexity of checking redundancy of CNF propositional formulae. In *17th European Conference on Artificial Intelligence (ECAI'02)*, pages 262–266, 2002.

[14] P. Liberatore. Redundancy in logic I: CNF propositional formulae. *Artificial Intelligence*, 163(2):203–232, 2005.

[15] P. Liberatore. Redundancy in logic II: 2CNF and Horn propositional formulae. *Artificial Intelligence*, 172(2–3):265–299, 2008.

[16] J. P. Marques-Silva and K. A. Sakallah. GRASP: A new search algorithm for satisfiability. In *International Conference on Computer-Aided Design (CAD'96)*, pages 220–227, Santa Clara (USA), 1996.

[17] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *13th Annual Symposium on Switching and Automata Theory (FOCS'72)*, pages 125–129, 1972.

[18] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *38th Design Automation Conference (DAC'01)*, pages 530–535, Las Vegas (USA), 2001.

[19] SAT competitions, http://www.satcompetition.org.

[20] H. Zeng and S. A. McIlraith. The role of redundant clauses in solving satisfiability problems. In *11th International Conference on Principles and Practice of Constraint Programming (CP'05)*, volume 3709 of *Lecture Notes in Computer Science*, page 873. Springer, 2005.

[21] L. Zhang and S. Malik. The quest for efficient boolean satisfiability solvers. In *18th International Conference on Automated Deduction (CADE'02)*, volume 2392 of *Lecture Notes in Artificial Intelligence*, pages 295–313. Springer, 2002.

[22] L. Zhang and S. Malik. Extracting small unsatisfiable cores from unsatisfiable boolean formula. In *6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919 of *Lecture Notes in Computer Science*, Portofino (Italy), 2003. Springer.