On the power of top-k undominated learnt clauses for Modern SAT Solvers

Jerry Lonlac, Engelbert Mephu Nguifo

On the power of top-k undominated learnt clauses for Modern SAT Solvers

Jerry Lonlac[a,b,*], Engelbert Mephu Nguifo[a]

[a] *University Clermont Auvergne, CNRS, LIMOS, F-63000 Clermont-Ferrand, France*
[b] *University Clermont Auvergne, CNRS, GEOLAB, F-63000 Clermont-Ferrand*

Abstract

Clause Learning is one of the most important components of a conflict driven clause learning (CDCL) SAT solver that is effective on industrial instances. Since the number of learned clauses is proved to be exponential in the worse case, it is necessary to identify the most relevant clauses to maintain and delete the irrelevant ones. As reported in the literature, several learned clauses deletion strategies have been proposed. However the diversity in both the number of clauses to be removed at each step of reduction and the results obtained with each strategy increase the difficulty to determine which criterion is better. Thus, the problem to select which learned clauses are to be removed during the search step remains very challenging. In this paper, we propose a novel approach to identify the most relevant learned clauses without favoring or excluding any of the proposed measures, but by adopting the notion of dominance relationship among those measures. Our approach bypasses the problem of results diversity and reaches a compromise between the measures assessments. Furthermore, the proposed approach also avoids another non-trivial problem which is the number of deleted clauses at each reduction of the learned clause database.

Keywords: Satisfiability, Clause Learning, CDCL SAT Solver, Dominance Relationship.

*Corresponding author
Email address: jerry.lonlac_konlac@uca.fr (Jerry Lonlac)
URL: www.isima.fr/~lonlac/ (Jerry Lonlac)

1. Introduction

The SAT problem, i.e., the problem of checking whether a Boolean formula in conjunctive normal form (CNF) is satisfiable or not, is central to many domains in computer science and artificial intelligence including constraint satisfaction problems (CSP), automated planning, non-monotonic reasoning, VLSI correctness checking, etc. Today, SAT has gained a considerable audience with the advent of a generation of solvers able to solve large instances encoding real-world problems. These solvers, often called *modern SAT solvers* [1, 2] or CDCL (Conflict Driven Clause Learning) SAT solvers have been shown to be very efficient at solving real-world SAT instances. They are built by integrating four major components into the classical Davis, Putnam, Logemann and Loveland procedure, commonly called DPLL [3]: lazy data structures [1], activity-based variable selection heuristics (VSIDS-like) [1], restart policies [4], and clause learning [5, 1]. Although a nice combination of these components contributes to improve the efficiency of modern SAT solvers [6], clause learning is known as the most important component [7].

The global idea of clause learning is that during the unit propagation process, when a current branch of the search tree leads to a conflict, moderns SAT solvers learn a conflict clause that helps unit propagation to discover one of the implications missed at an earlier level. This conflict clause expresses the causes of the conflict and is used to prune the search space. Clause learning, also known in the literature as Conflict Driven Clause Learning (CDCL), refers now to the most known and used First UIP learning scheme, first integrated in the SAT solver Grasp [8] and efficiently implemented in zChaff [1]. Most of the SAT solvers integrate this strong learning scheme. Since at each conflict, CDCL solvers learn a new clause that is added to the learned clauses database, and the number of learned clauses is proved to be exponential in the worse case, it is necessary to remove some learned clauses to maintain a database of reasonable size. Therefore, removing too many clauses can make learning inefficient, and keeping too many clauses also can alter the efficiency of unit propagation.

Managing the learned clauses database was the subject of several studies [1, 8, 2, 9, 10, 11]. These strategies were proposed with the objective to maintain a learned clause database of reasonable size by eliminating clauses deemed irrelevant to the subsequent search. The general principle of these strategies is that, at each conflict, an activity is associated to the learned clauses (static strategy). Such heuristic-based activity aims to weight each clause according to its relevance to the search process. In the case of dynamic strategies, such clauses activities are dynamically updated. Although all the learned clause deletion strategies proposed in the literature are shown to be empirically efficient, identifying the most relevant clause to maintain during the search process remains a challenging task.

Other works from the literature on the clause learning component have instead focused on the minimization of the learned clause. These works aim mostly at reducing the number literals from learned clauses [12, 13, 14, 15, 16]. In this way, more recently, in [17], the authors propose a new in-processing learned clause minimization approach able to remove redundant literals from learned clauses for CDCL solvers. This approach is based on Boolean constraint propagation, or more precisely unit (clause) propagation, which is time-consuming on large instances. The integration of this approach in the state-of-the art best SAT solvers allows to solve a large number of additional instances coming from the hard combinatorial and application categories of the 2014 and 2016 SAT competitions. The SAT solver `Maple_LCM_Dist` winner of the last 2017 SAT competition on the Main Track instances implements this minimization approach.

By considering the impact of clause learning on the practical resolution of SAT instances, our motivation in this work comes from the observation that the use of different relevant-based deletion strategies gives different performances.

Our goal is to take advantage of several relevant learned clauses deletion strategies by seeking a compromise between them through a dominance relationship.

In this paper, we integrate a user-preference point of view in the SAT process.

To this end, we integrate into the SAT process the idea of skyline queries [18], dominant patterns [19], undominated association rules [20] in order to learn clauses in a threshold-free manner. Such queries have attracted considerable attention due to their importance in multi-criteria decision making. Given a set of clauses, the skyline set contains the clauses that are not dominated by any other clause.

Skyline processing does not require any threshold selection function, and the formal property of domination satisfied by the skyline clauses gives to the clauses a global interest with semantics easily understood by the user. This skyline notion has been developed for database and data mining applications, however it was unused for SAT purposes. In this paper, we adapt this notion to the learned clauses management process.

In our previously related work [21], we propose to search at each reduction step of the learned clauses database, the current reference learned clause [21] (top-1 undominated learned clauses in this paper) according to a set of relevant measures and to delete all the learned clauses dominated by this clause. In this paper, we extend this idea by considering instead k current reference learned clause i.e the top-k undominated learned clauses at each cleaning step and delete all the learned clauses dominated by at least one of the top-k undominated learned clauses.

The paper is organized as follows. We first present some effective relevant-based learned clauses deletion strategies used in the literature. Then, our learned clauses deletion strategy based on the dominance relationship between different strategies is presented in Section 4. Finally, before the conclusion, experimental results demonstrating the efficiency of our approach are presented.

2. Around SAT Problem

Let $\mathcal{L}$ be a propositional language of formulas $\mathcal{F}_{\mathcal{L}}$ built in the standard way, using usual logical connectives $\neg$, $\vee$, $\wedge$, $\rightarrow$, $\Rightarrow$, $\Leftrightarrow$, and a set of propositional variables. Using linear Tseitin encoding [22], any Boolean formula $f \in \mathcal{F}_{\mathcal{L}}$ can

be translated to $CNF(f)$, a formula in conjunctive normal form. A CNF formula is a conjunction ($\wedge$) of clauses, where a clause is a disjunction ($\vee$) of literals. A literal is a propositional variable (l) or a negated propositional variable ($\neg l$). The literals l and $\neg l$ are called complementary literals. A unit clause is a clause containing only one literal (called unit literal). An empty clause, noted $\bot$, is interpreted as false (unsatisfiable), whereas an empty CNF formula, noted $\top$, is interpreted as true (satisfiable). A CNF formula Φ can also be seen as a set of clauses, and a clause as a set of literals. The size of the formula Φ corresponds to the value $\sum_{c \in \Phi} |c|$ where $|c|$ is the number of literals in the clause c. An assignment $\mathcal{I}$ of is a function which associates a value $\mathcal{I}(\mathrm{x}) \in \{false, true\}$ to some of the variables of Φ. $\mathcal{I}$ is complete if it assigns a value to every variable of Φ, and partial otherwise. A model of a formula is an assignment that makes the formula *true*. The SAT problem consists in determining if a Boolean formula expressed in CNF admits a model or not.

SAT is the NP-complete decision problem for which the largest amount of research effort has been expended for the development of sophisticated algorithms with highly-optimized implementations. Most of the SAT algorithms are highly complex and empirical studies allows to assess and compare their performance. An SAT competition organized each year allows both an objective assessment of these SAT algorithms and a promotion of new SAT solvers. However, one solver can be better than others at solving some SAT instances from a given class, but dramatically worse on other instances. There is no solver effective on all SAT instances class, different solvers perform best on different instances. Thus, from this observation, rather than following the traditional approach of choosing the best solver for a given class of instances, some works on SAT build the portfolios algorithm [23, 24, 25, 26, 27, 28, 29] that selects solvers on a per-instance basis using empirical hardness models for runtime prediction [30]. These empirical hardness models are usually built using machine learning techniques. A predictor of an algorithm's runtime on a given problem instance use the set of features of the instance and the algorithms past performance [31, 32]. In order to do this, instances are grouped into classes based on their features,

and the best algorithm is calculated for each class. Thereafter, given an input instance, its features are computed and it is assigned to a class (using the prediction model previously built), and it is solved by the corresponding solver assigned to that class. In recent years, there have been some works to better characterize the SAT instances in order to effectively classify them [33, 34]. In fact, the set of features used to build the classifiers of SAT instances plays a crucial role. In [35], the authors use some structure features of industrial SAT instances to build some classifiers of industrial SAT families of instances. The effectiveness of these classifiers is measured by comparing them to other sets of SAT features commonly used in portfolio SAT solving approaches. In [36], it is proved that the ratio between the run-times needed by a CDCL solver and by a random-specialized solver is related to the scale-free structure of the boolean formula.

The international SAT competition series stimulate the development of efficient implementations leading to increasingly complex solvers with several number of parameters. These parameters allow solvers to be customized to a particular family of SAT instances. However, in the SAT competitions, solvers are run using a single default parameter setting supplied by the authors for all benchmark instances in a given track, this poses a problem for a practical use of SAT who only cares about performance on one particular application and can invest some time into tuning solver parameters for this application. So, a new Configurable SAT solver Competition (CSSC) [37] has been designed in order to evaluate solver performance for each SAT instance after having configured its parameters. The CSSC takes into account the fact that effective algorithm configuration procedures can automatically customize solvers for a given distribution of benchmark instances. More precisely, for each type of SAT instances and each SAT solver, an automated fixed-time offline configuration phase determines parameter settings of solver optimized for high performance on this type of SAT instances. Then, the performance of solver on the type of instance is evaluated with these settings, and the solver with the best performance wins.

3. On the learned clauses database management strategies

In this section, we present some efficient learned clauses relevance measures exploited in the most SAT solvers of the literature.

The most popular CDCL SAT solver *Minisat* [2] considers as relevant the clauses the most involved in recent conflict analysis and removes the learned clauses whose involvement in recent conflict analysis is marginal. Another strategy called *LBD* for *Literal Block Distance* was proposed in [9]. LBD based measure is also exploited by most of the best state-of-the-art SAT solvers (*Glucose, Lingeling [38]*) and whose efficiency has been proved empirically. LBD based measure uses the number of different levels involved in a given learned clause to quantify the quality of the learned clauses. Hence, the clauses with smaller LBD are considered as more relevant. In [10], a new dynamic management policy of the learned clauses database is proposed. It is based on a dynamic freezing and activation principle of the learned clauses. At a given search state, using a relevant selection function based on progress saving (PSM), it activates the most promising learned clauses while freezing irrelevant ones. In [11], a new criterion to quantify the relevance of a clause using its backtrack level called BTL for BackTrack Level was proposed. From experiments, the authors observed that the learned clauses with small BTL values are used more often in the unit propagation process than those with higher BTL values. More precisely, the authors observed that the learned clauses with BTL value less than 3 are always used much more than the remaining clauses. Starting from this observation, and motivated by the fact that a learned clause with smaller BTL contains more literals from the top of the search tree, the authors deduce that relevant clauses are those allowing a higher backtracking in the search tree (having small BTL value). More recently, several other learned clauses database strategies were proposed in [39, 40]. In [39], the authors explore a number of variations of learned clause database reduction strategies, and the performance of the different extensions of *Minisat* solver integrating their strategies is evaluated on the instances of the SAT competitions 2013/2014 and compared against other state-of-the-art

SAT solvers (*Glucose, Lingeling*) as well as against default *Minisat*. From the performances obtained in [39], the authors have shown that size-bounded learning strategies proposed more than fifteenth years ago [8, 41, 42] is not over and remains a good measure to predict the quality of learned clauses. They show that adding randomization to size bounded learning is a nice way to achieve controlled diversification, allows to favor the short clauses, while maintaining a small fraction of large clauses necessary for deriving resolution proofs on some SAT instances. In [40], the authors use the community structure of industrial SAT instances to identify a set of highly useful learned clauses. They show that augmenting a SAT instance with the clauses learned by the solver during its execution does not always mean to make easy the resolution of the instance. However, the authors show that augmenting the formula with a set of clauses based on the community structure of the formula improves the performance of the solver in many cases. The different performances obtained by each strategy suggests that the question on how to predict efficiently the "best" learned clauses is still open and deserves further investigation.

On the other hand, it is important to note that the efficiency of most of these state-of-the-art learned clauses management strategies heavily depends on the cleaning frequency and on the amount of clauses to be deleted each time. Generally, all the CDCL SAT solvers using these strategies exactly delete half of the learned clauses at each learned clauses database reduction step. For example, the CDCL SAT solver *Minisat* [2] and *Glucose* [9] delete half of the learned clauses at each cleaning. Therefore, the efficiency of this amount of learned clauses to delete (e.g the half) at each cleaning step of the learned clauses database has not been demonstrated theoretically, but instead experimentally. For our knowledge, there are not many studies in the literature on how to determine the amount of clauses to be deleted each time. This paper proposes an approach to identify the relevant learned clauses during the resolution process without favoring any of the best reported relevant measures and which frees itself of the amount of clauses to be removed at each time: the amount of learned clauses to delete corresponds at each time to the number of learned clauses

dominated by one particular learned clause of the set of the current learned clauses which is called in the following sections, the reference learned clause.

4. Detecting undominated learned Clauses

We present now our learned clauses relevant measure based on dominance relationship. We first motivate this approach with a simple example, and then propose an algorithm allowing to identify the relevant clauses with some technical details.

4.1. Motivating example

Let us consider the following relevant strategies: *LBD* [9], *SIZE* (which consider as relevant the clause of the short size) and the relevant measure use by *minisat* [2] that we denote here *CVSIDS*. Suppose that we have in the learned clauses database, the clauses c_1, c_2 and c_3 with:

- $SIZE(c_1) = 8$, $LBD(c_1) = 3$, $CVSIDS(c_1) = 1e^{100}$;
- $SIZE(c_2) = 6$, $LBD(c_2) = 5$, $CVSIDS(c_2) = 1e^{200}$;
- $SIZE(c_3) = 5$, $LBD(c_3) = 4$, $CVSIDS(c_3) = 1e^{300}$.

The question we ask is the following: which one is relevant? In [9], the authors consider the clause c_1 which has the most smallest LBD measure as the most relevant. In contrast, the authors of [39] and [43] prefer the clause c_3 while the preference of the authors of *Minisat* [2] leads to the clause c_3. Our approach copes with the particular preference at one measure by finding a compromise between the different relevant measures through the dominance relationship. Hence, for the situation described above, only the clause c_2 is irrelevant because it is dominated by the clause c_3 on the three given measures.

4.2. Formalization

During the search process, the CDCL SAT solvers learn a set of clauses which are stored in the learned clauses database Δ, $\Delta = \{c_1, c_2, ..., c_n\}$. At each cleaning step, we evaluate these clauses with respect to a set $\mathcal{M} = \{m_1, m_2, ..., m_k\}$ of relevant measures. We denote $m(c)$ the value of the measure m for the clause

c, $c \in \Delta$, $m \in \mathcal{M}$. Since the evaluation of learned clauses varies from a measure to another one, using several measures could lead to different outputs (relevant clauses with respect to a measure). For example, if we consider the motivating example, c_1 is the best clause with respect to the LBD measure whereas it is not the case according to the evaluation of $SIZE$ measure which favors c_3. This difference of evaluations is confusing for any process of learned clauses selection. Hence, we can utilize the notion of dominance between learned clauses to address the selection of relevant ones. Before, formulating the dominance relationship between learned clauses, we need to define it at the level of measure values. To do that, we define dominance value as follows:

Definition 1 (dominance value). *Given a learned clauses relevant measure m and two learned clauses c and c', we say that $m(c)$ dominates $m(c')$, denoted by $m(c) \succeq m(c')$, iff $m(c)$ is preferred to $m(c')$. If $m(c) \succeq m(c')$ and $m(c) \neq m(c')$ then we say that $m(c)$ strictly dominates $m(c')$, denoted $m(c) \succ m(c')$.*

Definition 2 (dominance clause). *Given two learned clauses c, c', the dominance relationship according to the set of learned clauses relevant measures $\mathcal{M}$ is defined as follows:*

- *c dominates c', denoted $c \succeq c'$, iff $m(c) \succeq m(c')$, $\forall m \in \mathcal{M}$.*
- *If c dominates c' and $\exists m \in \mathcal{M}$ such that $m(c) \succ m(c')$, then c stritly dominates c' and we note $c \succ c'$.*

To discover the relevant learned clauses a naive approach consists in comparing each clause with all other ones. However, the number of learned clauses is proved to be exponential which makes pairwise comparisons costly. In the following, we show how to overcome this problem by defining at each cleaning step of learned clauses database, a particular learned clause that we call here *current Reference Learned Clause (in short RLC)* which is an undominated clause of Δ according to the set of learned clauses relevant measures $\mathcal{M}$. At each cleaning step, all the learned clauses dominated by the *current Reference Learned Clause*

(in short RLC) will be considered as the irrelevant learned clauses and thus deleted from the learned clauses database.

To define *current Reference Learned Clause*, we need a new relevant measure based on all the learned clauses relevant measures of $\mathcal{M}$. We call this new measure *Degree of compromise*, in short *DegComp* defines as follows:

Definition 3 (Degree of compromise). *Given a learned clause c, the degree of compromise of c with respect to the set of learned clauses relevant measures $\mathcal{M}$ is defined by $DegComp(c) = \frac{\sum_{i=1}^{|M|} \widehat{m_i(c)}}{|M|}$, where $\widehat{m_i(c)}$ corresponds to the normalized value of the clause c on the measure m_i.*

Following the same idea, and in order to avoid keep too much learned clauses in the learned clauses database at each reduction step, we propose a variant of our dominance strategy that considers k *Reference Learned Clauses (in short k-RLC)* instead one that we called here 1-*RLC*. More precisely, k-*RLC* strategy is defined as follows: at each reduction step, we delete from the learned clauses database all the clauses dominated by at least one of the k first undominated learned clauses according to the set of learned clauses relevant measures.

In fact, in practice, measures are heterogeneous and defined within different scales. For example the values of the learned clauses relevant measures in [2] are very high, in exponential order while the values of the relevant measures in [9] are smallest ones. Hence, in order to avoid that the measures with the higher values make marginal the measures with smallest values in the computation of the comprise degree of a given learned clauses, it is recommended to normalize the measures values. In our case here, we choose to normalize all the measures in the interval $[0, 1]$. More precisely, each value of measure $m(c)$ of any learned clause c must be normalized into $\widehat{m(c)}$ within $[0, 1]$. The normalization of a given measure m is performed depending on its domain and the statistical distribution of its active domain. We recall that the active domain of a measure m is the set of its possible values. It is worth mentioning, **the normalization of a measure does not modify the dominance relationship** between two given values. If we consider the learned clause c_1 given in the motivating example in the section

4.1, with its three values : $DegComp(c_1) = \frac{\widehat{CVSIDS}(c_1)+\widehat{LBD}(c_1)+\widehat{SIZE}(c_1)}{3}$, then, we have, $DegComp(c_1) = \frac{\frac{1}{1e^{100}}+\frac{3}{nVars()}+\frac{8}{nVars()}}{3}$, with $nVars()$ the number of variables of the Boolean formula.

Let us precise that, the value domain of all measures should be in accordance i.e either lower values are better or higher values are better. For example, if we consider the learned clause c_1, and the three relevant measures LBD, $SIZE$ and $CVSIDS$, we take $\frac{1}{CVSIDS(c_1)}$ value as the normalized value of the clause c_1 on the measure $CVSDIS$. In fact, $CVSIDS$ higher value is better, whereas LBD and $SIZE$ values are better. As $CVSIDS$ higher value is better, $\frac{1}{CVSDIS(c_1)}$ smaller value is so better also. In this way, the measures become comparable over clauses.

After giving the necessary definitions (current reference learned clause and Degree of compromise), the following lemma offers a swifter solution rather than pairwise comparisons, to find relevant clauses based on dominance relationship.

Lemma 1. *Let c be a learned clause having the minimal degree of compromise with respect to the set of learned clauses relevant measures $\mathcal{M}$, then c is an undominated clause.*

Proof 1. *Let c be a learned clause having the minimal degree of compromise with respect to the set of learned clauses relevant measures $\mathcal{M}$, we suppose that there exists a learned clause c' that strictly dominates c, which means that $\forall m \in \mathcal{M}$, $m(c') \succeq m(c)$ and $\exists m' \in \mathcal{M}$, $m'(c') \succ m'(c)$. Hence, we have $DegComp(c') < DegComp(c)$. The latter inequality contradicts our hypothesis, since c has the minimal degree of compromise with respect to $\mathcal{M}$.*

In our *k-RLC* dominance strategy, the k first undominates learned clauses are the k first learned clauses ranked in the increasing order of their degree of compromise. It is important to note that at each reduction step the k first undominates learned clauses are not necessary the k first learned clauses in the learned clauses database ranked in the increasing order of their degree of compromise.

Property 1. *Let $\mathcal{M}$ be the set of learned clauses relevant measures, $\forall c, c', c"$ three learned clauses, if $c \succ c'$ and $c' \succ c"$ then $c \succ c"$.*

Searching for all undominated clauses during each cleaning step can be time consuming, such that we only compute the undominated clauses with respect to the k reference learned clauses during each reduction step.

During the search process, at each cleaning step of the learned clauses database, We compare all the remaining learned clauses with each of the current reference learned clauses. Thus, for each learned clause, we perform at least 1 comparison and at most k comparisons with the k current reference learned clauses to determine if the clause is dominated and then deleted or if the clause is not dominated and therefore conserved.

Property 2. *Given a learned clause database $\Delta = \{c_1, c_2, ..., c_n\}$, k the number of the current Reference Learned Clauses, the time complexity of our k-RLC dominance approach is linear in the worst case.*

Proof 2. *Let $\Delta = \{c_1, c_2, ..., c_n\}$ be a a learned clause database, k the number of the current Reference Learned Clauses. For the k-RLC dominance approach, each learned clause is at most compared to the k first undominated learned clauses. So we can make at most $k \times n$ comparisons to detect all the learned clauses not dominated by none of the k first reference learned clauses.*

4.3. Algorithm

In this section, after presenting the general scheme of a deletion strategy of learned clauses ($reduceDB(\Delta)$) adopted by most of the reported solvers, we propose an algorithm allowing to discover relevant learned clauses by using dominance relationship.

Algorithm 1 depicts the general scheme of a learned clause deletion strategy ($reduceDB(\Delta)$). This algorithm first sorts the set of learned clauses according to the defined criterion and then deletes half of the learned clauses. In fact, this algorithm takes a learned clauses database of size n and outputs a learned

clauses database of size $n/2$. This is different from our approach which first sorts the set of learned clauses according to the degree of compromise of each clause and then removes all the learned clauses that are dominated by at least one of the first k undominated learned clauses.

Algorithm 2 depicts our learned clause deletion strategy. It is important to note that our dominance relationship approach does not remove learned clauses whose size (number of literals) and LBD are less than or equal to 2. These clauses are considered as more relevant and are maintained in the learned clauses database.

Algorithm 1: Deletion Strategy: `reduceDB` function

Input: Δ: The learned clauses database of size n

Output: Δ The new learned clauses database of size $n/2$

```
1  sortLearntClauses() ;            /* by the defined criterion */
2  limit = n/2;
3  ind = 0;
4  while ind < limit do
5  |  clause = Δ[ind] ;
6  |  if clause.size() > 2 and clause.lbd() > 2 then
7  |  |  removeClause() ;
8  |  else
9  |  |  saveClause() ;
10 |  ind++;
11 return Δ ;
```

Algorithm 2: `reduceDB-Dominance_Relationship`

Input: Δ: The learned clauses database; $\mathcal{M}$: a set of relevant measures; k: the number of reference learned clauses

Output: Δ The new learned clauses database

```
1  sortLearntClauses() ;  /* by degree of compromise criterion */
2  ind = 1;
3  j = 1;
4  undoC = 1 ;     /* the number of current undominated clauses */
5  while ind < |Δ| do
6      c = Δ[ind] ;                                  /* a learned clause */
7      if c.size() > 2 and c.lbd() > 2 then
8          cpt = 0 ;
9          while cpt < undoC and ¬dominates(Δ[cpt], Δ[ind], M) do
10             cpt++ ;
11         if cpt >= undoC then
12             saveClause() ;
13             j++ ;
14             undoC = min(k, j) ;          /* minimum between j and k */
15         else
16             removeClause() ;
17
18     else
19         saveClause() ;
20         j++ ;
21         undoC = min(k, j) ;              /* minimum between j and k */
22     ind++;
23 return Δ ;

24 Function dominates(cMin: a clause, c: a clause, M)
25 i = 0;
26 while i < |M| do
27     m = M[i] ;                                   /* a relevant measure */
28     if m(c) ⪰ m(cMin) then
29         return FALSE ;
30     i++;
31 return TRUE ;
```

Proposition 1. *The k Reference Learned Clauses considered by the algorithm 2 correspond to the k first undominated learned clauses ranked in the increasing order of their degree of compromise.*

Property 3. *Let* $\Delta = \{c_1, c_2, ..., c_n\}$ *be a learned clause database and* k *the number of the considered Reference Learned Clauses. Let* u *be the number of undominated learned clauses after applying the algorithm 2. We have* $k \leq u \leq |\Delta|$.

Proof 3. *Trivial by using the proposition 1.*

5. Experiments

For our experiments, we use three relevant measures for the dominance relationship to assess the efficiency of our approach. Notice that the user can choose to combine different other measures. We use SIZE [43], LBD [9] and $CVSIDS$ [2] measures. All these measures have been proved effective in the literature [2, 9, 39]. It is possible to use more relevant measures, but it should be noted that by adding a measure to $\mathcal{M}$, the number of relevant learned clauses maintained may decrease or increase. The decrease can be explained by the fact that a learned clause can be dominated with respect to a set of measures $\mathcal{M}$ and undominated with respect to $\mathcal{M}'$, such that $\mathcal{M} \subset \mathcal{M}'$. For example, if two learned clauses c and c' are undominated with respect to $\mathcal{M}$, there is a possibility that one of them dominates the other by removing one measure. The increase can be explained by the fact that a learned clause can be dominated with respect to $\mathcal{M}$ and undominated with respect to $\mathcal{M}'$. For example, consider a learned clause c which dominates another learned clause c' with respect to $\mathcal{M}$, by adding a measure m to $\mathcal{M}$, such that $m(c') \succ m(c)$, then c' is no longer dominated by c.

We run the SAT solvers on the 300 instances taken from the last SAT-RACE 2015 and on the 300 instances taken from the last SAT competition 2016. All

the instances are preprocessed by SatElite [44] before running the SAT solver. The experiments are made using Intel Xeon quad-core machines with 32GB of RAM running at 2.66 Ghz. For each instance, we used a timeout of 1 hour of CPU time for the SAT-RACE, and 10000s for the SAT Competition. We integrate our approach in *Glucose* and made a comparison between the original solver and the one enhanced with the new deletion learned clause strategy using dominance relationship called *k-RLC-Glucose*. To determine the best upper bound size k, we run *k-RLC-Glucose* with $k = 1$, 3, 5 and 6.

5.1. Number of solved instances and CPU time

Table 1 presents results on SAT-RACE-2015. We use the source code of *Glucose* 3.0 with the measure LBD (written *LBD-Glucose* or *Glucose* in what follows). We then replace LBD by each of the other measures : $SIZE$-*Glucose* that considers the shortest clauses as the most relevant, $CVSIDS$-*Glucose* that maintains the learned clauses most involved in recent conflict analysis, $RAND$-*Glucose* that randomly deletes learned clauses and finally our proposal *k-RLC-Glucose* that deletes from the learned clauses database all the clauses dominated by the k first undominated learned clauses ranked in the increasing order of their degree of compromise. Table 1 shows the comparative experimental evaluation of the four measures as well as $Minisat$ 2.2. In the second column of Table 1, we give the total number of solved instances (#Solved). We also mention, the number of instances proven satisfiable (#SAT) and unsatisfiable (#UNSAT) in parenthesis. The third column shows the average CPU time in seconds (total time on solved instances divided by the number of solved instances). On the SAT-RACE 2015, our approach *k-RLC-Glucose* is more efficient than the others in terms of the number of solved instances (see also Figure 1). In fact the original solver *Glucose* solves 236 instances while it is enhanced with our dominance approach as 10 (respectively 12) more instances are solved by *k-RLC-Glucose* for $k = 3$ (respectively $k = 6$). In fact, solving such additional numbers of instances is clearly significant in practical SAT solving. The $CVSIDS$-*Glucose* solver solves 4 more instances than *Glucose* 3.0. $Minisat$ 2.2 solves only 209

instances.

As randomization of clause deletion sometimes pay [39], we quantify the performance gap between our strategy and the random deletion-based one called here *RAND-Glucose*. The solver *RAND-Glucose* is obtained as follows: at each conflict, the activity of the learned clause c is set to random value $irand(random\_seed, |V_{\mathcal{F}}|)$, where $irand(random\_seed, |V_{\mathcal{F}}|)$ return a number between 0 and $|V_{\mathcal{F}}|$ and $V_{\mathcal{F}}$ denotes the set of variables occurring in the boolean formula $\mathcal{F}$. We used exactly the random function of *Glucose* with the same *random_seed* to allow reproducible results. We can see as reported in the table 1 that *RAND-Glucose* is the worst solver among the five solvers, it solves 174 instances, 72 (respectively 74) instances of less than our solver *k-RLC-Glucose* for $k = 3$ (respectively $k = 6$). This shows the interest of our dominance relationship approach on the instances of the SAT-RACE 2015.

Solvers	#Solved (#SAT - #UNSAT)	Average Time
Minisat 2.2	209 (134 - 75)	585.19 s
RAND-Glucose	174 (99 - 75)	608.82 s
SIZE-Glucose	230 (131 - 99)	533.86 s
CVSIDS-Glucose	240 (140 - 100)	622.23 s
LBD-Glucose	236(136 - 100)	**481.66 s**
1-*RLC-Glucose*	238 (138 - 100)	**481.72 s**
3-*RLC-Glucose*	**246 (144 - 102)**	523.46 s
5-*RLC-Glucose*	245 (144 - 101)	542.29 s
6-*RLC-Glucose*	**248 (145 - 103)**	532.05 s

Table 1: Comparative evaluation on SAT-RACE-2015.

Figure 1 shows the cumulated time results i.e. the number of instances (x-axis) solved under a given amount of time in seconds (y-axis). This figure gives for each technique the number of solved instances (*#instances*) in less than t seconds. It confirms the efficiency of our dominance relationship approach. From this figure, we can observe that *k-RLC-Glucose* (for the values $k = 3, 6$)

is generally faster than all the other solvers, even if the average running time of *LBD-Glucose* is slightly better (see Table 1). Although 3-*RLC-Glucose* needs additional time to compute the dominance relationship, the quality of the remained clauses on SAT-RACE helps to improve the time needed to solved the instances.

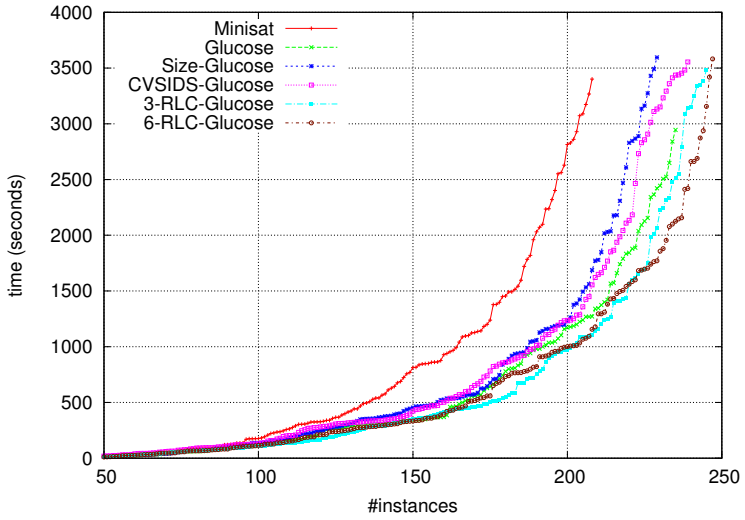

Figure 1: **Evaluation on SAT-RACE-2015**

Table 2 shows 6 instances of the SAT-RACE 2015 solved by our approach best version 6-*RLC-Glucose* in less than 2200 seconds but not solved by *LBD-Glucose*, *SIZE-Glucose*, nor *CVSIDS-Glucose* in 3600*seconds*. The time used to solve those instances may also explain the increase of the average running time of 6-*RLC-Glucose*. In addition we also find that there is none instance solved by all the other solvers and not solved by 6-*RLC-Glucose* (as detailed later). This shows on the one hand that the application of dominance between different relevant measures does not degrade the performance of all the solvers but instead takes advantage of the performance of each relevant measure, considering the SAT-RACE 2015 dataset.

Table 3 presents results on the instances of the SAT Competition 2016. Here *LBD-Glucose* and *CVSIDS-Glucose* solve the same number of instances. Our

Instances	LBD	SIZE	CVSIDS	6-RLC
jgiraldezlevy.2200.9086.08.40.8	-	-	-	**80.57 s**
jgiraldezlevy.2200.9086.08.40.149	-	-	-	**393.51 s**
jgiraldezlevy.2200.9086.08.40.2	-	-	-	**1599.86 s**
manthey_DimacsSorterHalf_37_3	-	-	-	**1763.42 s**
manthey_DimacsSorter_37_3	-	-	-	**1770.69 s**
hwmcc10-*-pdtvisns3p00-tseitin	-	-	-	**2146.03 s**

Table 2: Instances solved by 6-*RLC-Glucose* and not solved by the others on SAT-RACE 2015.

solver k-*RLC-Glucose* remains competitive and solves more instances than the original solver *Glucose* for $k = 3, 6$. The figure 2 presents the cumulated time results on the instances of the SAT competition 2016. It comes out from this second dataset that 3-*RLC-Glucose* is slightly faster than *LBD-Glucose*. *LBD-Glucose* is more efficient than the other solvers. A more fine analysis of the figure 2 shows that 3-*RLC-Glucose* is generally more faster on the instances solved in less than 3000 seconds while *LBD-Glucose* is generally more faster on the instances solved between 3000 seconds and 5000 seconds. This can be explained by the fact that sometimes during the resolution step, very few learned clauses are dominated, thereby our solver keeps too much in the learned clauses database.

This outcome gives credit to the NO FREE Lunch theorem [45]. We also think that the aggregated function may not be unique for all the datasets, such that it is necessary to explore the efficient combination of the preferred measures.

RAND-Glucose solves only 121 instances, 48 instances less than 3-*RLC-Glucose*.

Table 4 shows 6 instances of the SAT Competition 2016 solved by our approach best version 3-*RLC-Glucose* among which 5 instances are solved in less than 5000 seconds but not solved by *LBD-Glucose*, *SIZE-Glucose*, nor *CVSIDS-Glucose*. This confirms as on the instances of the SATRACE 2015

Solvers	#Solved (#SAT - #UNSAT)	Average Time
Minisat 2.2	138 (65 - 73)	1194.85 s
RAND-Glucose	121 (56 - 65)	1120.99 s
SIZE-Glucose	156 (67 - 89)	1396.73 s
CVSIDS-Glucose	165 (67 - **98**)	1368.99 s
LBD-Glucose	165 (68 - 97)	**1142.33 s**
1-*RLC-Glucose*	156 (64 - 92)	1227.62 s
3-*RLC-Glucose*	**169 (71 - 98)**	1297.36 s
5-*RLC-Glucose*	165 (68 - 97)	1352.46 s
6-*RLC-Glucose*	167 (70 - 97)	1439.21 s

Table 3: Comparative evaluation on SAT-Competition-2016.

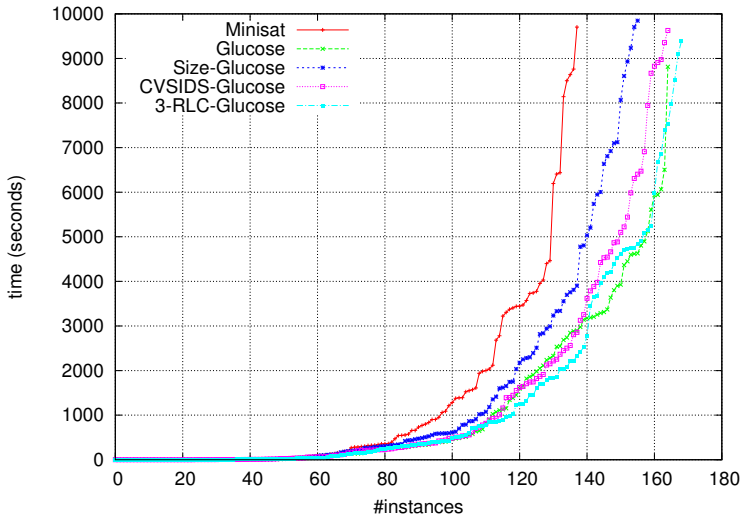

Figure 2: **Evaluation on SAT competition 2016**

that our dominance approach does not degrade the performance of all the solvers but instead takes advantage of the performance of each relevant measure.

Instances	LBD	SIZE	CVSIDS	3-RLC
barman-pfile08-032.sas.ex.15	-	-	-	**5.01 s**
partial-10-19-s	-	-	-	**1248.47 s**
ak128modasbg2asisc	-	-	-	**1021.06 s**
par32-3-c	-	-	-	**2206.07 s**
gss-24-s100	-	-	-	**4753.82 s**
eq.atree.braun.13.unsat	-	-	-	**6683.69 s**

Table 4: Instances solved by 3-*RLC-Glucose* and not solved by the others on SAT-Competition-2016.

5.2. Common solved instances

In Table 5, the intersection between two relevant measures gives the number of common instances solved by each measure. For example, LBD and $SIZE$ solved 219 instances in common, while 235 instances are solved by LBD and 6-RLC. We can see than our approach solves the largest number of instances in common with each of the aggregated measures. More precisely, the number of common instances solved with another measure is lower than the number of common instances solved with our approach.

Measures	LBD	SIZE	CVSIDS	6-RLC
LBD	236			**235**
SIZE	219	230		**224**
CVSIDS	233	221	240	**234**
6-RLC				**248**

Table 5: Common solved Instances from SAT-RACE-2015.

The table 6 gives the result of the intersection of the results between two relevant measures on the instances of the SAT competition 2016. For a given relevant measure, we can see that the number of common instances solved with another measure is less than or equal to the number of common instances solved with our approach k-*RLC-Glucose* for $k = 3$.

Measures	LBD	SIZE	CVSIDS	3-RLC
LBD	165			**161**
SIZE	153	156		**153**
CVSIDS	161	151	165	**161**
3-RLC				**169**

Table 6: Common solved Instances from SAT competition 2016.

To get more details, Table 7 gives the number of instances commonly solved by the considered relevant measures on the instances of the SAT-RACE-2015. This table allows to see the number of common instances solved by one, two, three or four measures. For example, there are 218 common instances solved by the four deletion strategies, while 43 instances are not solved by none of them. We can observe that 0, 3, 4, and 6 are the number of instances solved alone by respectively LBD and $CVSIDS$, $SIZE$ and 6-RLC. Moreover, there is no instance solved by the three strategies (LBD, $SIZE$ and $CVSIDS$) and not solved by our approach 6-RLC.

Measures		6-RLC		¬6-RLC	
		CVSIDS	¬CVSIDS	CVSIDS	¬CVSIDS
LBD	SIZE	218	1	**0**	0
	¬SIZE	14	2	1	0
¬ LBD	SIZE	1	4	2	4
	¬SIZE	1	**6**	3	43

Table 7: Detailed of common instances with SAT-RACE 2015.

The table 8 gives more details on the number of instances commonly solved by the considered relevant measures on the instances of the SAT competition 2016. From this table, we observe that 150 common instances solved by the four deletion strategies and 123 instances not solved by none of them. It is important to note on these instances of the SAT competition 2016 no instance is solved by the three strategies (LBD, $SIZE$ and $CVSIDS$) and not solved

by our 3-RLC approach while 6 instances are solved by 3-RLC and not solved by none other measure.

Measures		3-RLC		¬3-RLC	
		CVSIDS	¬CVSIDS	CVSIDS	¬CVSIDS
LBD	SIZE	150	1	**0**	2
	¬SIZE	10	0	1	**1**
¬ LBD	SIZE	1	1	0	**1**
	¬SIZE	0	**6**	**3**	123

Table 8: Detailed of common instances with SAT competition 2016.

5.3. Combined measures

Table 9 gives on the number of instances of the SAT-RACE 2015 solved with our dominance approach w.r.t the measures used in the dominance relations. From this table, we can see that the number of instances solved by using our approach (two or three measures in the dominance) is always greater than the number of instances solved by using each measure alone. We also note that, the number of instances solved by using two measures (instead of three) in the dominance relationship is always lower than the number of instances solved (248) by using three measures.

Measures	LBD	SIZE	CVSIDS	6-*RLC-Glucose*
LBD	236			
SIZE	**238**	230		
CVSIDS	**245**	**246**	240	
6-*RLC-Glucose*				**248**

Table 9: Combining two measures on SAT-RACE-2015.

The results of the table 10 confirm the results of the table 9 of the SAT RACE-2015 in the sense that the number of instances of the SAT competition 2016 solved by using two measures (160 by using SIZE and LBD measures,

162 by using LBD and CVSIDS measures, 161 by using SIZE and CVSIDS measures) instead of three in our dominance approach is always lower than the number of instances solved by using three measures (169).

Measures	LBD	SIZE	CVSIDS	3-*RLC-Glucose*
LBD	165			
SIZE	160	156		
CVSIDS	162	161	165	
3-*RLC-Glucose*				**169**

Table 10: Combining two measures on SAT competition 2016.

5.4. Percentage of deleted clauses

During our experiments, we compute at each reduction step of instance resolution, the percentage of deleted clauses i.e the number of dominated clauses (which are removed) over the total number of learned clauses during this step. This allows to obtain an average percentage of deleted learned clauses per solved instance. By taking all the solved instances, we compute for each version of our solver the average of the average percentage of deleted learned clauses `(AverageDeleted)`. We give also the average of the maximum percentage of deleted learned clauses `(AverageDeletedMax)` and the average of the minimum percentage of deleted learned clauses `(AveragDeletedMin)`. The results are presented in Table 11 (respectively Table 12) for the instances of the SAT-RACE 2015 (respectively SAT competition 2016). We can see that on the instances of the SAT-RACE 2015, our best version 6-*RLC-Glucose* deletes in average 37% of learned clauses at each reduction step, with a standard deviation of 16%.

Figures 3 and 4 plot for each solved instance of the SAT-RACE 2015 (X-axis), the average percentage of deleted learned clauses (red curve with left-Y-axis) against respectively the total resolution time (overall solving time) and the average resolution time (green curve with right-Y-axis). For each solved instance, the average resolution time is obtained by dividing its total resolution

Solvers	AverageDeleted	AverageDeletedMax	AverageDeletedMin
1-*RLC-Glucose*	0.25	0.77	0.05
3-*RLC-Glucose*	0.33	0.78s	0.07
5-*RLC-Glucose*	0.36	0.78	0.07
6-*RLC-Glucose*	0.37	0.79	0.07

Table 11: Average of the average (respectively maximum and minimum) percentage of deleted learned clauses on SAT-RACE-2015.

Solvers	AverageDeleted	AverageDeletedMax	AverageDeletedMin
1-*RLC-Glucose*	0.30	0.70	0.08
3-*RLC-Glucose*	0.34	0.71	0.08
5-*RLC-Glucose*	0.37	0.71	0.09
6-*RLC-Glucose*	0.37	0.71	0.09

Table 12: Average of the average (respectively maximum and minimum) percentage of deleted learned clauses on SATcompetition 2016.

time by the number of reductions made before solving the instance. We look these statistics on our best version 6-*RLC-Glucose*.

On the Figures 3 and 4, the (red) curve of the average percentages of deleted learned clauses exhibits a high variation of the percentages of reduction from 0.15 to 0.95, with an average value equals to 0.37 and a standard deviation of 0.16. It comes out from this figure that the average percentage of deleted learned clauses is less than 50% on 217 instances among 248 solved instances. Our current strategy which uses k ($k = 6$ for these statistics) undominated learned clauses at each step is satisfactory wrt the running time, even if it can be possible to extend this stategy to a reduction with many undominated clauses. The curve of the average percentages of deleted learned clauses also shows 17 instances having the average percentage of deleted learned clauses equal to 0. These 17 instances correspond to the instances solved by the solver without having to reduce the learned clauses database.

Figure 3 shows on the one hand, the instances whose resolution times are small but with a high average percentage of deleted learned clauses, and on the other hand, the instances whose resolution times are high but with a low average

percentage of deleted learned clauses. In contrast, on the Figure 4 where we use the average resolution time instead of the total resolution time, we remark that the instances with the high average percentage of deleted learned clauses have generally the small average resolution times. However, this figure shows some instances with the small average percentage of deleted learned clauses and whose the average resolution times are small.

This clearly shows that the number of deleted learned clauses at each reduction step is important, but is not the only component that impacts the resolution time. Other key components of modern CDCL SAT solver such as the restart policies [4] and the activity-based variable selection heuristics [1] also have an influence on the resolution time.

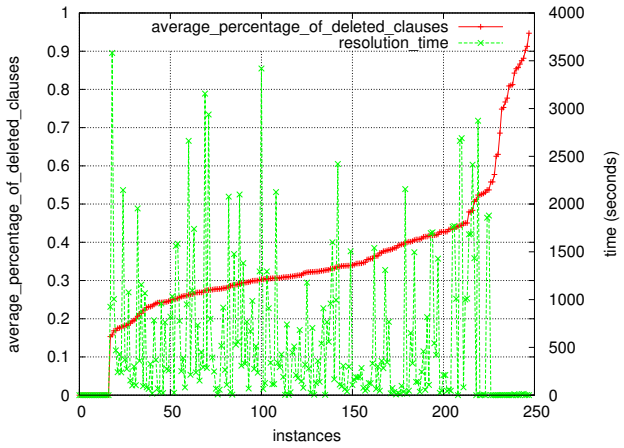

Figure 3: **Average percentage of deleted clauses and resolution time for each instance of the SAT RACE-2015 with** 6-*RLC-Glucose* **solver .**

The figures 5 and 6 respectively show the same results that those of figures 3 and 4 but on the instances of the SAT competition 2016 with our 3-*RLC-Glucose* solver. These figures clearly confirm the remark know of the literature and already made on the instances of the SAT-RACE 2015 namely that the resolution time of a SAT instance does not only depends on the number of deleted learned clauses at each reduction step of the learned clauses database.

However, the analysis of the different Figures 3, 4, 5 and 6 bring the follow-

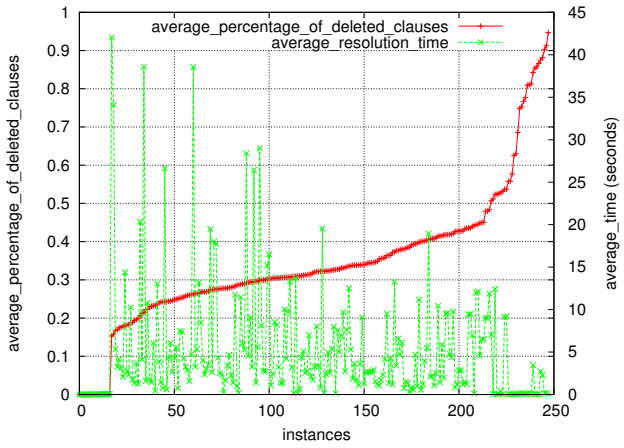

Figure 4: **Average percentage of deleted clauses and average resolution time for each instance of the SAT RACE-2015 with** 6-*RLC*-*Glucose* **solver.**

ing knowledge: for a quickly solving, some instances need the deletion of high number of learned clauses at each reduction step, in contrast, others instances need the deletion of small number of learned clauses. The information is that, we should find the strategies allowing the solver to identify alone at each reduction step both the number of learned clauses to delete and the different clauses to delete. This perfectly fits with the ideas developed in this paper.

For the instances of the SAT competition 2016, the curve of the average percentages of deleted learned clauses exhibits a very high variation of the percentages of reduction from 0.09 to 0.89, with an average value equals to 0.34 and a standard deviation of 0.18. We can observe from the figures 5 and 6 that about 85% of the instances are solved by deleting less than 50% of learned clauses at each reduction step. Compared to the resolution of the instances of the SAT-RACE 2015, our approach keeps more much of learned clauses at each reduction step. However the fact that the total number of clauses that were deleted has changed (37% of the learned clauses removed on the SAT-RACE 2015 and 34% removed on the SAT competition 2016) is not the cause of the small improvement on the instances of the SAT competition 2016, but it is well due to our dominance criterion. In fact, on the instances of the SAT-RACE

2015 (respectively SAT competition 2016) changing the remove half to remove 37% or less than 37% of the learned clauses database (respectively 34% or less than 34%) does not give a similar achievement (see the figures 7 and 8).

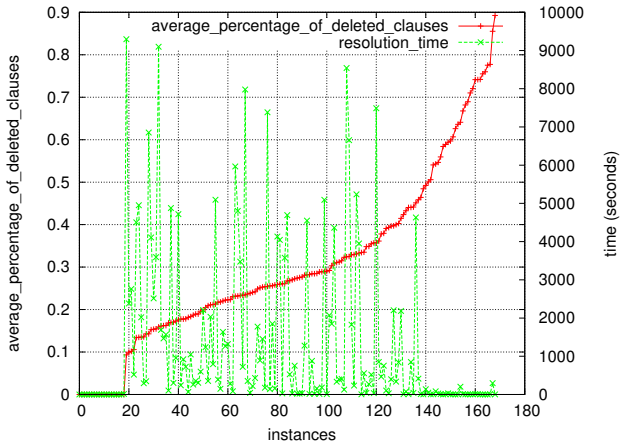

Figure 5: **Average percentage of deleted clauses and resolution time for each instance of the SAT competition 2016 with** 3-*RLC*-*Glucose* **solver.**

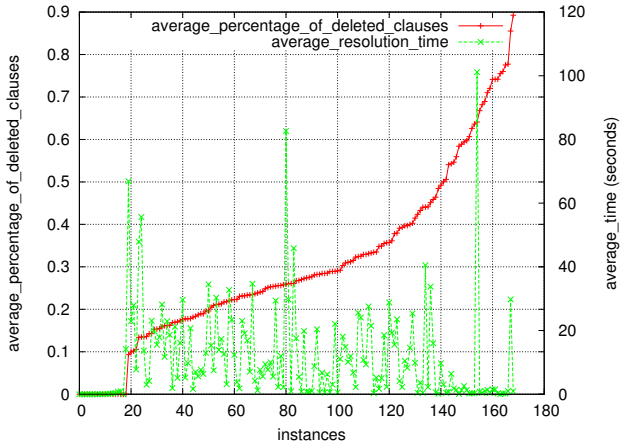

Figure 6: **Average percentage of deleted clauses and average resolution time for each instance of the SAT competition 2016 with** 3-*RLC*-*Glucose* **solver.**

The figure 7 gives the number of instances solved by each version of *Glucose* solver integrating the deletion strategies LBD, $SIZE$ and $CVSIDS$ depending on the number of learned clauses to be deleted at each cleaning step of the

learned clauses database during the resolution process.

The goal of this experiment is to see the impact of the quantity of learned clauses to be deleted at each cleaning step on the performance of solver.

As our proposed approach removes approximately an average 36% of learned clauses at each cleaning step, we run each of the three solvers with the quantity of learned clauses to be deleted set to 10%, 20%, 30% and 40% instead of 50%.

The figure 7 (respectively figure 8) shows that for each of the three deletion strategies (LBD, $SIZE$ and $CVSIDS$), even by removing less than fifty percent of the learned clauses at each reduction, the performances of *Glucose* solver are not improved on the SAT-RACE 2015 instances (respectively SAT competition 2016). This clearly shows the benefit of our dominance relationship on these instances.

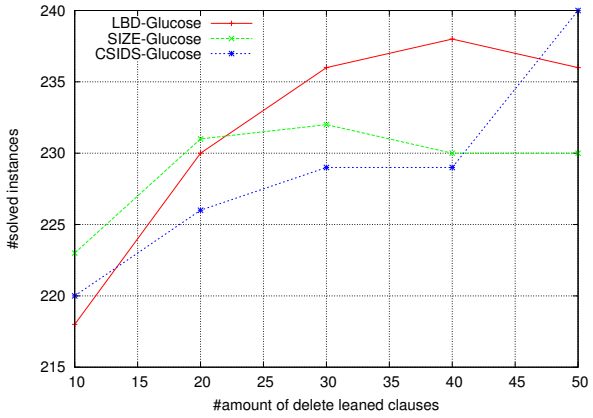

Figure 7: **Number of solved instances per percentage of deleted learned clauses on the instances of the SAT-RACE-2015**

It should be noted that if the solver *Glucose* keeps all the learned clauses (no learned clauses deleted) throughout the resolution process, it solves only 203 instances on the SAT-RACE 2015 instances in 1 hour (33 instances less than the original solver and 45 instances less than the solver integrating our dominance approach). On the instances of the 2016 SAT competition, keeping all the learned clauses during the resolution process, the solver glucose solves

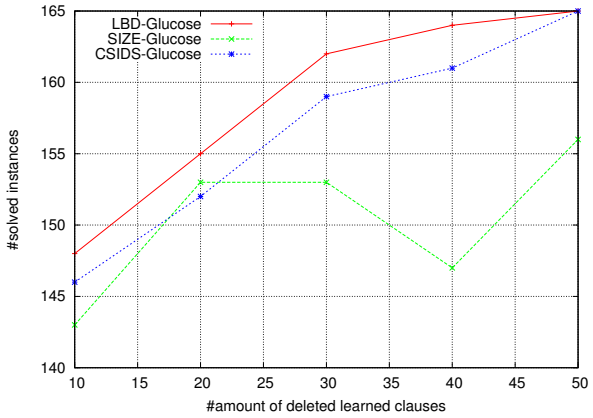

Figure 8: **Number of solved instances per percentage of deleted learned clauses on the instances of the SAT competition 2016**

only 131 in 10000 seconds (34 instances less than the original solver and 33 instances less than the solver integrating our approach).

This confirms the need to eliminate certain learned clauses (those deemed irrelevant) during the resolution process, and otherwise the interest of the learned clauses removal problem in CDCL SAT solvers.

6. Conclusion and future works

In this paper, we propose an approach that addresses the learned clauses database management problem. We have shown that the idea of dominance relationship between relevant measures is a nice way to take advantage of each measure. At each reduction step of the learned clauses database, we propose to select the top-k current undominated learned clauses (the k first undominated Reference Learned Clauses) according to a set of relevant learned clauses measures, and to delete all the clauses dominated by at least one of the top-k current undominated learned clauses. This approach is not hindered by the abundance of relevant measures which has been the issue of several works. The proposed approach avoids another non-trivial problem which is the amount of learned clauses to be deleted at each reduction step of the learned clauses database by

dynamically determining the number of learned clauses to delete at each cleaning step. A general algorithm for our approach is proposed and evaluated on the instances of the the SAT-RACE 2015 and the SAT competition 2016. The experimental results show that exploiting the dominance relationship improves the performance of CDCL SAT solver on these instances. The improvements are more significant on the instances of the SAT-RACE 2015. For the case of SAT-Competition where our approach achieve a small improvement, we must explore the effects of other key components of CDCL SAT solvers on our dominance approach. The instances categories might also be an issue which should be explored.

To the best of our knowledge, this is the first time that dominance relationship has been used in the satisfiability domain to improve the performance of a CDCL SAT solver. Our approach opens interesting perspectives. In fact, any new relevant measure of learned clauses can be integrated into the dominance relationship.

Acknowledgements

The authors would like to thank Auvergne-Rhône-Alpes region and European Union for their financial support through the European Regional Development Fund (ERDF). The authors would also like to thank CRIL (Lens Computer Science Research Lab) for providing them computing server and the authors of *Glucose* solver for making available the source code of their solver.

References

[1] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: Engineering an efficient sat solver, in: 38th Design Automation Conference (DAC), 2001, pp. 530–535.

[2] N. Eén, N. Sörensson, An extensible sat-solver, in: SAT, 2003, pp. 502–518.

[3] M. Davis, G. Logemann, D. W. Loveland, A machine program for theorem-proving., Communications of the ACM 5 (7) (1962) 394–397.

[4] C. P. Gomes, B. Selman, H. A. Kautz, Boosting combinatorial search through randomization, in: AAAI/IAAI, 1998, pp. 431–437.

[5] J. P. M. Silva, K. A. Sakallah, GRASP: A search algorithm for propositional satisfiability, IEEE Trans. Computers 48 (5) (1999) 506–521.

[6] H. Katebi, K. A. Sakallah, J. P. M. Silva, Empirical study of the anatomy of modern sat solvers, in: SAT, 2011, pp. 343–356.

[7] K. Pipatsrisawat, A. Darwiche, On the power of clause-learning sat solvers with restarts, in: (CP'09), 2009, pp. 654–668.

[8] J. P. M. Silva, K. A. Sakallah, Grasp - a new search algorithm for satisfiability, in: International Conference on Computer-Aided Design (ICCAD), 1996, pp. 220–227.

[9] G. Audemard, L. Simon, Predicting learnt clauses quality in modern sat solvers, in: IJCAI, 2009, pp. 399–404.

[10] G. Audemard, J.-M. Lagniez, B. Mazure, L. Sais, On freezing and reactivating learnt clauses, in: SAT, 2011, pp. 188–200.

[11] L. Guo, S. Jabbour, J. Lonlac, L. Sais, Diversification by clauses deletion strategies in portfolio parallel SAT solving, in: ICTAI, 2014, pp. 701–708.

[12] A. Biere, Preprocessing and inprocessing techniques in SAT, in: Hardware and Software: Verification and Testing - 7th International Haifa Verification Conference, HVC 2011, Haifa, Israel, December 6-8, 2011, Revised Selected Papers, 2011, p. 1.

[13] A. Wotzlaw, A. van der Grinten, E. Speckenmeyer, Effectiveness of pre- and inprocessing for cdcl-based SAT solving, CoRR abs/1310.4756.

[14] N. Sörensson, A. Biere, Minimizing learned clauses, in: Theory and Applications of Satisfiability Testing, 12th International Conference, SAT, Swansea, UK, June 30 - July 3, 2009. Proceedings, 2009, pp. 237–243.

[15] P. Beame, H. A. Kautz, A. Sabharwal, Towards understanding and harnessing the potential of clause learning, J. Artif. Intell. Res. 22 (2004) 319–351.

[16] S. Jabbour, J. Lonlac, L. Sais, Extending resolution by dynamic substitution of boolean functions, in: IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI , Athens, Greece, November 7-9, 2012, pp. 1029–1034.

[17] M. Luo, C. Li, F. Xiao, F. Manyà, Z. Lü, An effective learnt clause minimization approach for CDCL SAT solvers, in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI, Melbourne, Australia, August 19-25, 2017, pp. 703–711.

[18] S. Börzsönyi, D. Kossmann, K. Stocker, The skyline operator, in: Proceedings of the 17th International Conference on Data Engineering, April 2-6, Heidelberg, Germany, 2001, pp. 421–430.

[19] A. Soulet, C. Raïssi, M. Plantevit, B. Crémilleux, Mining dominant patterns in the sky, in: ICDM, 2011, pp. 655–664.

[20] S. Bouker, R. Saidi, S. Ben Yahia, E. Mephu Nguifo, Mining undominated association rules through interestingness measures, International Journal on Artificial Intelligence Tools 23 (04) (2014) 1460011.

[21] J. Lonlac, E. Mephu Nguifo, Towards learned clauses database reduction strategies based on dominance relationship, CoRR abs/1705.10898.

[22] G. Tseitin, On the complexity of derivations in the propositional calculus, in: H. Slesenko (Ed.), Structures in Constructives Mathematics and Mathematical Logic, Part II, 1968, pp. 115–125.

[23] C. P. Gomes, B. Selman, Algorithm portfolios, Artif. Intell. 126 (1-2) (2001) 43–62.

[24] S. Kadioglu, Y. Malitsky, M. Sellmann, K. Tierney, ISAC - instance-specific algorithm configuration, in: ECAI - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, Proceedings, 2010, pp. 751–756.

[25] Y. Malitsky, A. Sabharwal, H. Samulowitz, M. Sellmann, Non-model-based algorithm portfolios for SAT, in: Theory and Applications of Satisfiability Testing - SAT - 14th International Conference, Ann Arbor, MI, USA, June 19-22. Proceedings, 2011, pp. 369–370.

[26] M. Nikolic, F. Maric, P. Janicic, Simple algorithm portfolio for SAT, Artif. Intell. Rev. 40 (4) (2013) 457–465.

[27] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, Y. Shoham, A portfolio approach to algorithm selection, in: IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003, p. 1542.

[28] L. Xu, F. Hutter, H. H. Hoos, K. Leyton-Brown, Satzilla: Portfolio-based algorithm selection for SAT, J. Artif. Intell. Res. 32 (2008) 565–606.

[29] L. Xu, F. Hutter, H. H. Hoos, K. Leyton-Brown, : The design and analysis of an algorithm portfolio for SAT, in: Principles and Practice of Constraint Programming, 13th International Conference, CP, Providence, RI, USA, September 23-27, Proceedings, 2007, pp. 712–727.

[30] F. Hutter, L. Xu, H. H. Hoos, K. Leyton-Brown, Algorithm runtime prediction: Methods & evaluation, Artif. Intell. 206 (2014) 79–111.

[31] K. Leyton-Brown, E. Nudelman, Y. Shoham, Learning the empirical hardness of optimization problems: The case of combinatorial auctions, in: Principles and Practice of Constraint Programming - CP, 8th International

Conference, Ithaca, NY, USA, September 9-13, 2002, Proceedings, 2002, pp. 556–572.

[32] E. Nudelman, K. Leyton-Brown, H. H. Hoos, A. Devkar, Y. Shoham, Understanding random SAT: beyond the clauses-to-variables ratio, in: Principles and Practice of Constraint Programming - CP, 10th International Conference, Toronto, Canada, September 27 - October 1, Proceedings, 2004, pp. 438–452.

[33] C. Ansótegui, J. Giráldez-Cru, J. Levy, The community structure of SAT formulas, in: Theory and Applications of Satisfiability Testing - SAT - 15th International Conference, Trento, Italy, June 17-20. Proceedings, 2012, pp. 410–423.

[34] Z. Newsham, V. Ganesh, S. Fischmeister, G. Audemard, L. Simon, Impact of community structure on SAT solver performance, in: Theory and Applications of Satisfiability Testing - SAT - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL, Vienna, Austria, July 14-17. Proceedings, 2014, pp. 252–268.

[35] C. Ansótegui, M. L. Bonet, J. Giráldez-Cru, J. Levy, Structure features for SAT instances classification, J. Applied Logic 23 (2017) 27–39.

[36] J. Giráldez-Cru, J. Levy, A modularity-based random SAT instances generator, in: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, 2015, pp. 1952–1958.

[37] F. Hutter, M. Lindauer, A. Balint, S. Bayless, H. H. Hoos, K. Leyton-Brown, The configurable SAT solver challenge (CSSC), Artif. Intell. 243 (2017) 1–25.

[38] A. Biere, Lingeling and friends entering the sat challenge 2012, in: A. Balint, A. Belov, A. Diepold, S. Gerber, M. Jarvisalo, , C. S. (editors) (Eds.), Proceedings of SAT Challenge 2012: Solver and Benchmark

Descriptions, vol. B-2012-2 of Department of Computer Science Series of Publications B, University of Helsinki, 2012, pp. 33–34.

[39] S. Jabbour, J. Lonlac, L. Sais, Y. Salhi, Revisiting the learned clauses database reduction strategies, CoRR abs/1402.1956.

[40] C. Ansótegui, J. Giráldez-Cru, J. Levy, L. Simon, Using community structure to detect relevant learnt clauses, in: SAT, 2015, pp. 238–254.

[41] R. J. Bayardo, D. P. Miranker, A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem, in: In Proceedings of the Thirteenth National Conference on Artificial Intelligence, 1996, pp. 298–304.

[42] R. J. Bayardo, Jr., R. C. Schrag, Using CSP look-back techniques to solve real-world SAT instances, in: AAAI, 1997, pp. 203–208.

[43] E. Goldberg, Y. Novikov, Berkmin: A fast and robust sat-solver, Discrete Applied Mathematics 155 (12) (2007) 1549 – 1561.

[44] N. Eén, A. Biere, Effective preprocessing in sat through variable and clause elimination, in: SAT, 2005, pp. 61–75.

[45] D. Wolpert, W. G. Macready, No free lunch theorems for optimization, IEEE Trans. Evolutionary Computation 1 (1) (1997) 67–82.