Community and LBD-based Clause Sharing Policy for Parallel SAT Solving

Vincent Vallade[1], Ludovic Le Frioux[2], Souheib Baarir[1,3], Julien Sopena[1,4], Vijay Ganesh[5], and Fabrice Kordon[1]

[1] Sorbonne Université, CNRS, LIP6, UMR 7606, Paris, France
[2] LRDE, EPITA, Le Kremlin-Bicêtre, France
[3] Université Paris Nanterre, Nanterre, France
[4] Inria, DELYS Team, Paris, France
[5] University of Waterloo, Waterloo, ON, Canada

Abstract. Modern parallel SAT solvers rely heavily on effective clause sharing policies for their performance. The core problem being addressed by these policies can be succinctly stated as "the problem of identifying high-quality learnt clauses" that when shared between the worker nodes of parallel solvers results in improved performance than otherwise. The term "high-quality clauses" is often defined in terms of metrics that solver designers have identified over years of empirical study. Some of the more well-known metrics to identify high-quality clauses for sharing include clause length, literal block distance (LBD), and clause usage in propagation.

In this paper, we propose a new metric aimed at identifying high-quality learnt clauses and a concomitant clause-sharing policy based on a combination of LBD and community structure of Boolean formulas. The concept of community structure has been proposed as a possible explanation for the extraordinary performance of SAT solvers in industrial instances. Hence, it is a natural candidate as a basis for a metric to identify high-quality clauses. To be more precise, our metric identifies clauses that have low LBD and low community number as ones that are high-quality for applications such as verification and testing. The community number of a clause C measures the number of different communities of a formula that the variables in C span. We perform extensive empirical analysis of our metric and clause-sharing policy, and show that our method significantly outperforms state-of-the-art techniques on the benchmark from the parallel track of the last four SAT competitions.

1 Introduction

The encoding of complex combinatorial problems as Boolean satisfiability (SAT) instances has been widely used in industry and academy over the last few decades. From AI planning [19] to cryptography [26], modern SAT solvers have demonstrated their ability to tackle huge formulas with millions of variables and clauses. This is instinctively surprising since SAT is an NP-complete problem [12]. The high-level view of a SAT solver algorithm is the successive enumeration of all possible values for each variable of the problem until a solution is found or the unsatisfiability of the formula is concluded. What makes SAT solving applicable to large real-world problems is the conflict-driven clause learning (CDCL) paradigm [24].

During its search, a CDCL solver is able to learn new constraints, in the form of new implied clauses added to the formula, which allows it to avoid the exploration of large parts of the search space. In practice too many clauses are learnt and a selection has to be done to avoid memory explosion. Many heuristics have been proposed, in the sequential context, to reduce the database of learnt clauses. Such methods of garbage collection are usually quite aggressive and are based on measures, such as the literal block distance (LBD), whose aim is to quantify the usefulness of clauses [4].

The omnipresence of many-core machines has led to considerable efforts in parallel SAT solving research [7]. There exist two main classes of parallel SAT strategies: a cooperative one called divide-and-conquer [32] and a competitive one called portfolio [15]. Both rely on the use of underlying sequential worker solvers that might share their respective learnt clauses. Each of these sequential solvers has a copy of the formula and manages its own learnt clause database. Hence, not all the learnt clauses can be shared and a careful selection must be made in order for the solvers to be efficient. In state-of-the-art parallel solvers this filtering is usually based on the LBD metric. The problem with LBD is its locality, indeed a clause does not necessarily have the same LBD value within the different sequential solvers' context.

In this work we explore the use of a more global quality measure based on the community structure of each instance. It is well-known that SAT instances encoding real-world problems expose some form of modular structure which is implicitly exploited by modern CDCL SAT solvers. A recurring property of industrial instances (as opposed to randomly-generated ones) is that some variables are more constraint together (linked by more clauses). We say that a group of variables that have strong link with each other and few links with the rest of the problem form a community (a type of cluster over the variable-incidence graph of Boolean formulas). A SAT instance may contain tens to thousand of communities.

Contributions The primary contributions of this paper are the following:

- Based on statistics gathered during sequential SAT solver's executions on the benchmark from the SAT competition 2018, we study the relationship between LBD and community, and we analyse the efficacy of LBD and community as predictive metrics of the usefulness of newly learnt clauses.
- Based on this preliminary analysis, we propose to combine both metrics to form a new one and to use it to implement a learnt clause sharing policy in the parallel SAT solving context.
- We implement our new sharing strategy in the solver (`P-MCOMSPS` [22]) winner of the last parallel SAT competition in 2018, and evaluate our solver on the benchmark from the SAT competition 2016, 2017, 2018, and 2019. We show that our solver significantly outperforms competing solvers over this large and comprehensive benchmark of industrial application instances.

Paper structure The remainder of the paper is structured as follows: we introduce the basic concepts necessary to understand our work in Section 2. Section 3 presents preliminary analysis on LBD usage to provide intuition and motivate our work. Section 4 explores combination of LBD and COM measures to detect useful learnt clauses. Solvers and experimental results are presented in Section 5. Section 6 surveys some related works and Section 7 concludes this paper.

2 Preliminaries

This section introduces useful definitions, notations, and concepts that will be used in the remaining of the paper. It is worth noting that we consider the context of complete SAT solving, and thus we focus on the well-known conflict-driven clause learning (CDCL) algorithm [24]. For details on CDCL SAT algorithm we refer the reader to [9].

2.1 Boolean Satisfiability Problem

A *Boolean variable* is a variable that has two possible values: *true* or $false$. A *literal* is a Boolean variable or its negation (NOT). A *clause* is a finite disjunction (OR) of literals. A *conjunctive normal form (CNF) formula* is a finite conjunction (AND) of clauses. In the rest of the paper we use the term formula to refer to CNF formula. Moreover, clauses are represented by the set of their literals, and formulas by the set of their clauses.

For a given formula F, we define an *assignment* of variables of F as a function $\mathcal{A} : \mathcal{V} \rightarrow \{true, false\}$, where $\mathcal{V}$ is the set of variables appearing in F. A clause is satisfied when at least one of its literals is evaluated to *true*. A formula is satisfied if all its clauses are evaluated to *true*. A formula is said to be *satisfiable* (SAT) if there is at least one assignment that makes it *true*; it is reported *unsatisfiable* (UNSAT) otherwise. The Boolean satisfiability (SAT) problem consists in determining if a given formula is SAT or UNSAT.

2.2 Literal Block Distance

Literal block distance (LBD) [4] is a positive integer, that is used as a learnt clause quality metric in almost all competitive sequential CDCL-like SAT solvers. The LBD of a clause is the number of different decision levels on which variables of the clause have been assigned. Hence, the LBD of a clause can change over time and it can be (re)computed each time the clause is fully assigned.

2.3 Community

It is well admitted that real-life SAT formulas exhibit notable "structures", explaining why some heuristics such as VSIDS [27] or phase saving [30], for example, work well. One way to highlight such a structure is to represent the formula as a graph and analyze its shape. A structure of interest in this paper is the so-called *community structure* [1]. Let us have a closer look on this latter.

An *undirected weighted graph (graph, for short)* is a pair $G = (N, w)$, where N is the set of nodes of G, and $w : N \times N \to \mathbb{R}^+$ is the associated weight function which should be commutative.

The *variable incident graph (VIG)* [1] of a formula F is a graph whose nodes represent variables of F, and there exists an edge between two variables iff they shared appearance in at least a clause. Hence, a clause C results in $\binom{|C|}{2}$ edges. Thus, to give the same importance to each clause, edges have a weight : $w(x, y) = \sum_{\substack{C \in F \\ x,y \in C}} 1/\binom{|C|}{2}$.

The *community detection* of a graph is usually captured using the modularity metric [28]. The modularity function $\mathcal{Q}(G, P)$ (see equation 1), takes a graph G and a partition $P = \{P_1, \ldots, P_n\}$ of nodes of G. It evaluates the density of the connection of the nodes within a part relatively to the density of the entire graph *w.r.t.* a random graph with the same number of nodes and the same degree.

$$\mathcal{Q}(G, P) = \sum_{P_i \in P} \frac{\sum_{x,y \in P_i} w(x, y)}{\sum_{x,y \in N} w(x, y)} - \left(\frac{\sum_{x \in P_i} deg(x)}{\sum_{x \in N} deg(x)} \right)^2 \qquad (1)$$

The *modularity* of G is the maximal modularity, for any possible partition P of its nodes: $Q(G) = max\{\mathcal{Q}(G, P) \mid P\}$, and ranges over $[0, 1]$.

Computing the modularity of a graph is an NP-hard problem [11]. However, there exists greedy and efficient algorithms, returning an approximated lower bound for the modularity of a graph, such as the *Louvain method* [10].

In the remaining of the paper we use the Louvain method (with a precision $\epsilon = 10^{-7}$) to compute communities. Graphs we consider are VIG of formulas already simplified by the `SatElite` [13] preprocessor.

Community value of a clause We call COM the number of communities on which a clause span: we work on the VIG of the problem, so we consider communities of variables. Each variable belongs to a unique community (determined by the Louvain algorithm). To compute the COM of a clause, we consider the variables corresponding to the literals of the clause and we count the number of distinct communities represented by these variables.

3 Measures and Intuition

Our main goal is to improve the overall performances of parallel SAT solving. One way to do that is to study the quality/impact of shared information between the underling SAT engines (sequential SAT solvers) in a parallel strategy.

Since determining, a priori, the usefulness of information is already a challenging task in a sequential context, we can easily imagine the hardness of this guessing in the parallel setting.

In almost all competitive parallel SAT solvers, the sharing is limited to particular forms of learnt clauses (*unit* clauses, clauses with an $LBD \leq i$ [8,22], double touched clauses [5]). We propose here to focus our study on LBD and evaluate the impact of sharing clauses with particular values.

3.1 Sequential SAT Solving, Learnt Clauses and LBD

The starting point of our analysis is to evaluate the usage of learnt clauses in the two main components of sequential solvers, namely, the *unit propagation* and the *conflict analysis* procedures. Performing this analysis in a sequential setting makes perfect sense since parallel solvers launch multiple sequential solvers.

To conduct this study, we run `MapleCOMSPS` [23] on the main track benchmark from the SAT competition in 2018[6] with a 5000 seconds timeout. Fig. 1 depicts our observations. The x-axis shows the percentage of the mean number of learnt clauses (considering all learnt clauses of the whole benchmark). The y-axis corresponds to the cumulative usage percentage. Hence, the curve with dots depicts, for different LBDs (from 1 to 9), the usage of these clauses in unit propagation. The curve with triangles highlights the same information but for conflict analysis.[7]

In both curves, the key observation concerns the inflection located around LBD=4. The impact of clauses with LBD ≤ 4 can be considered as very positive: 60% of usage in unit propagation, and 40% in conflict analysis, while representing only 3% of the total number of learnt clauses. Moreover, clauses with an LBD > 4 do not bring a significant added value when considering their quantities.

Based on these results, 4 appears to be a good LBD value for both limiting the rate of shared clauses and maximizing the percentage of tracked events in a parallel context.

3.2 A First Parallel Sharing Strategy

To assess our previous observation, we developed a strategy implementing an *LBD-based clauses sharing*: clauses learnt with an LBD below a predetermined threshold are shared.

We then operated this strategy with LBD=4, but also with the surrounding values (LBD=3 and LBD=5).

Our strategy has been integrated into `Painless` [21][8] using `MapleCOMSPS` as a sequential solver engine and a portfolio parallelization strategy.

The different solvers we compare are:

- `P-MCOMSPS`, a portfolio SAT solver, winner of the parallel track of the SAT competition 2018, is used as a reference. It implements a sharing strategy based on incremental values for LBD [22].

[6] http://sat2018.forsyte.tuwien.ac.at

[7] On the contrary of this qualitative clause study, benchmarks presented in Sections 3.2 and 5.2 do not have any logs.

[8] A framework to implement parallel solvers, thus allowing a fair comparison between strategies.

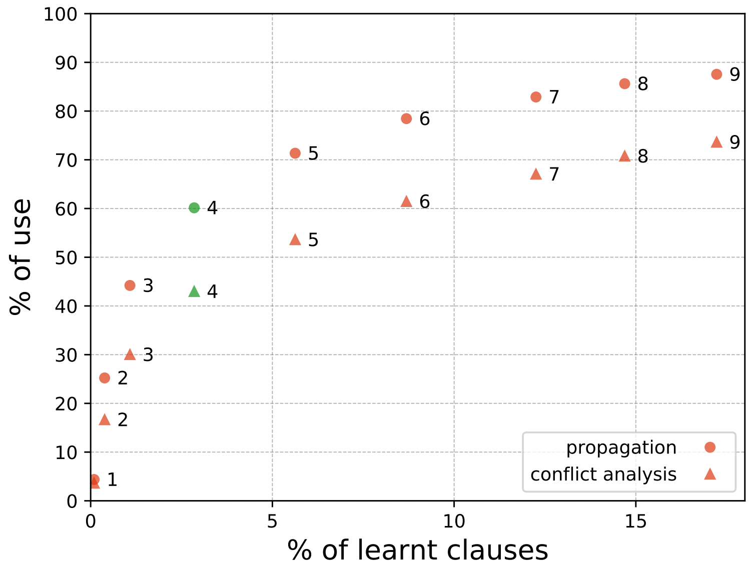


Fig. 1: Usage of learnt clause considering LBD

– `P-MCOMSPS-L`$\langle n \rangle$, our new portfolio solver, with LBD $\leq n$.

All solvers were processed on a 12-core Intel Xeon CPU at 2.40GHz and 62GB of RAM running Linux 4.9. The solvers have been compiled with the version 9.2.1 of *GCC*. They have been launched with 12 threads, a 61GB memory limit, and a wall clock-time limit of 5000 seconds (as for the SAT competitions). Table 1 presents our measures on the SAT 2017, SAT 2018, and SAT 2019 competition benchmarks. [9] The shaded cells indicate which solver has the best results for a given benchmark. It shows that the strategy based on an LBD ≤ 4 is not as

[9] In Table 1, PAR-k is the penalized average runtime, counting each timeout as k times the running time cutoff. The used value for k in the yearly SAT competition is 2.

Solvers	2017		2018		2019	
	PAR-2	# solved instances	PAR-2	# solved instances	PAR-2	# solved instances
`P-MCOMSPS`	**355h42**	**237**	430h13	258	**380h03**	**273**
`P-MCOMSPS-L3`	361h27	234	420h53	263	393h04	269
`P-MCOMSPS-L4`	356h44	**237**	**411h14**	**265**	391h38	269
`P-MCOMSPS-L5`	369h27	229	415h52	264	389h27	269

Table 1: Performances comparison for several values of LBD (3, 4, and 5). `P-MCOMSPS` is used as a reference.

efficient as we could expect. In particular, we note a large instability depending on the sets of instances to be treated.

We believe these results are due to the fact that the LBD is too related to the local state of the solver engines. The intuition we investigate in this paper is that the LBD metrics must be strengthened with more global information.

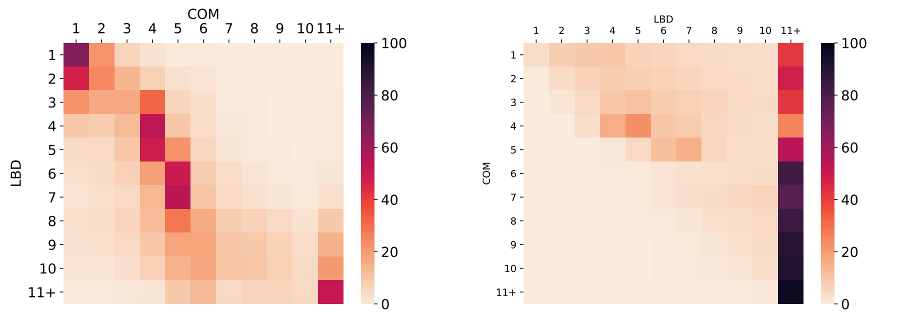
4 Combining LBD and Community for Parallel SAT Solving

As previously stated, we need a metric independent of the local state of a particular solver engine. Structural information about the instance to be solved can be useful. For instance, in a portfolio solver, structural information can be shared among the solvers working on the same formula.

In this paper, we focus on the community structure exhibited by (industrial) SAT instances. The metrics (COM, defined in Sec. 2.3) derived from this structure has been proven to be linked with the LBD in [29]. Besides, it has been used to improve the performances of sequential SAT solving via a preprocessing approach in [2].

This section shows that communities are good candidates to provide the needed global information. To do so, we use the same protocol as the one used in the previous section: *(i)* studying data on sequential SAT solving to exhibit good candidates for the parameter values, and then *(ii)* check its efficiency in a parallel context.

We completed the logs extracted from the sequential experiments presented in Section 3 by information on communities and studied the whole package as follows.



(a) COM distribution for various LBD (b) LBD distribution for various COM

Fig. 2: Heatmap showing the distribution between COM and LBD

4.1 LBD versus Communities

First, we studied the relationship between LBD and COM values thanks to two heatmaps (see Fig. 2). The left part (Fig. 2a) shows, for each LBD value, the distribution of COM values. The right part (Fig. 2b) shows, for each COM value, the LBD values distribution.

For example, in Fig. 2a, we observe that $\approx 65\%$ of the clauses with LBD=1 span on one community (COM=1), $\approx 20\%$ of them span on two communities (COM=2), etc.

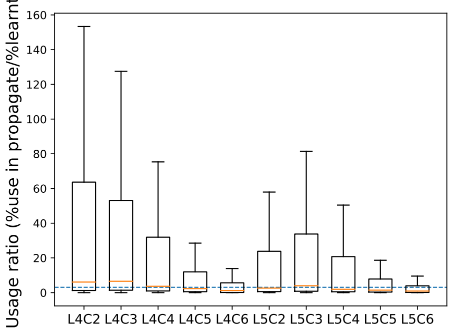
From these figures, we can conclude two important statements:

- from Fig. 2a, warm zones on the diagonal for low LBD values indicate a a sort of correlation between the LBD and COM values (a result that has been already presented in [29]). Looking closer, we observe that a significant part of the clauses does not follow this correlation: the warm zones remain below 65%, and mainly range below 25%. Hence, the COM metrics appears to be a good candidate to refine the clauses already selected using LBD;
- from Fig. 2b, the COM values are almost uniformly distributed all over the LBD values. We conclude that using COM as the only selecting metrics is misleading (we assessed this with several experiments with parallel solvers that are not presented in this paper).

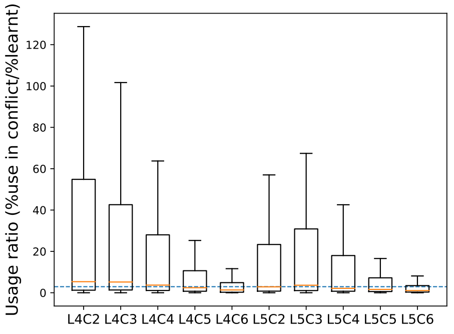
4.2 Composing LBD and Communities

As previously stated, COM is a good additional criterion to LBD. We thus need to discover the good couples of values that maximize the usage of shared clauses, while maintaining a reasonable size.

First, we note from Fig. 1 that clauses with LBD ≤ 3 represent 1.1% of the total clauses for a percentage of use rising up to 44.2%. Thus, there is no benefit to further filter those clauses. Besides, this first set of clauses is not



(a) Usage ratio propagate



(b) Usage ratio conflict

Fig. 3: Efficiency of LBD and COM combination. The blue dotted line corresponds to the average and the orange lines to the median.

sufficient since Table 1 reports that the parallel solver for LBD=3 never wins (this is mainly due to a lack of shared clauses). Therefore, we look for the set of clauses that can be added to LBD $\leq$ 3 to improve performances. Based on the observations made in Section 3, we believe the best candidate is a subset of LBD=4 or LBD=5. To identify the good couple(s), we reused our previous experimentation protocol and tracked data for both LBD and COM.

To capture the usefulness of learnt clauses, we define the *usage ratio* as follows: the ratio of the percentage of use (propagation or conflict analysis) to the percentage of learnt clauses (among those of a given formula). For instance, in a given formula, a clause with a *usage ratio* of 10 is used 10 times more than the average use of all learnt clauses. By extension, the *usage ratio of a set of clauses* is the average of the usage ratio of all its clauses.

The resulting data is displayed as box-plot diagrams in Fig. 3. These diagrams integrate the distribution of the usage ratio for all the formulas in the SAT competition 2018. A box-plot denoted L$\langle x\rangle$C$\langle y\rangle$ corresponds to the set of clauses with LBD=x and COM=y.

Our immediate observation (it confirms our intuition) is that, for a fixed value of LBD, the usage ratio varies heavily considering different COM values. Secondly, we discern that clauses with LBD=4 have a global better usage ratio than those with LBD=5. The third, and critical, observation allows us to extract the best promising configurations. Actually, L4C2 and L4C3 configurations have the most impressive usage ratio: in 25% (the 3^{rd} quartile) of the treated instances, clauses within these configurations have a usage ratio greater than 50 in the unit propagation (Fig. 3a), 40 in the conflict analysis (Fig. 3b) and extends to very high values (up to 150 in propagation and 130 in conflict analysis). Moreover, the median of propagation usage ratio for L4C2 and L4C3 (6.0 and 6.5, respectively) are twice as big as the mean of the entire LBD = 4 and LBD = 5 boxes (equal to 2.9 and noted with a dashed line in both figures).

4.3 Community Based Filtering

From this study, we can conclude that we can use the community structure as a filter for those clauses that have been already selected using an LBD threshold.

Practically, we propose the following strategy: sharing all the clauses with an LBD $\leq$ 3 (without any community limit) as well as those with an LBD $\leq$ 4 and a COM $\leq$ 3. Indeed, the former set of clauses is small while being very useful. Whereas, to select the clauses with COM = 3 among the set of clauses with LBD = 4 should allow a higher usage ratio while keeping the sharing at a reasonable ratio.

To verify this assertion, and validating the effectiveness of the chosen filter threshold (COM $\leq$ 3), we extend the study presented in Section 3.1. Thus, Fig. 4 and 5 take up the same points of Fig. 1, while separating propagation and conflict analysis results (to increase readability). To those points, the new figure shows, by triangles, the usage ratio for different filter values.

As expected, the point COM = 3 (green triangle) is at the inflection of the curves. Consequently, selecting this set of clauses allows us to reach a better

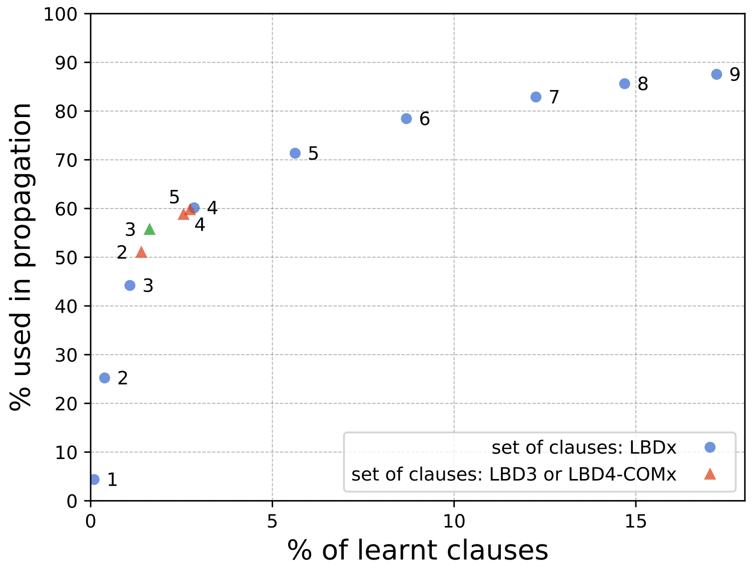


Fig. 4: Propagation usage of learnt clause considering LBD

usage ratio, close to the unfiltered LBD $\leq$ 4, while preserving a comparable number of clauses with LBD $\leq$ 3. This convinces us that this metrics should lead to increased performance in parallel SAT solving.

The following section verifies these measures in practice.

5 Derived Parallel Strategy and Experimental Results

This section first describes parallel SAT solvers we have designed to evaluate our strategy, as well as the associated experimental protocol. It then presents and discusses our experimental results.

5.1 Solvers and Evaluation Protocol

As in Section 3.2, we use the solver `P-MCOMSPS` as a reference to validate our proposal. The only difference between the original solver and the newly developed resides in their sharing strategies. These are as follows:

– `P-MCOMSPS`: the same strategy, based on incremental values for LBD.
– `P-MCOMSPS-L4C3`: only learnt clauses with an LBD $\leq$ 3 or LBD = 4 and a COM value $\leq$ 3 are shared.

In `P-MCOMSPS-L4C3`, a special component (called *sp*) is dedicated to compute the community structure (using the Louvain algorithm). Meanwhile, the remaining components execute the CDCL algorithm to solve the formula, and share clauses with an LBD $\leq$ 3. As noted in Section 4.2, sharing all these clauses

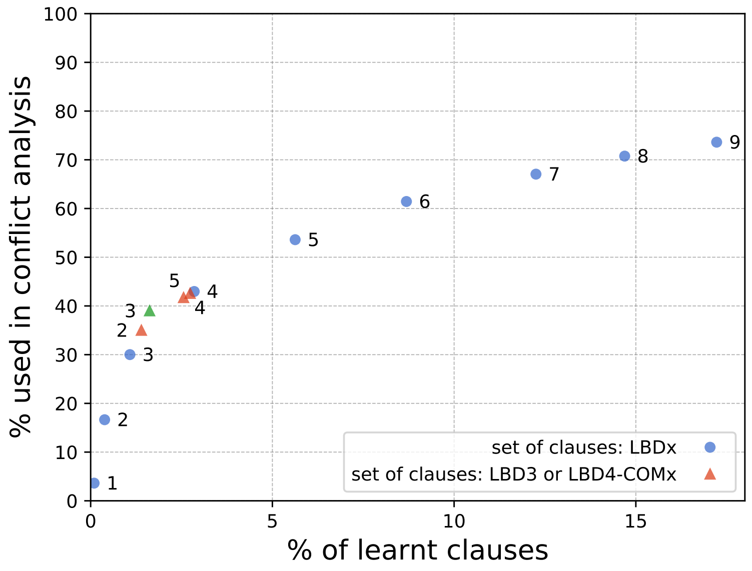


Fig. 5: Conflict analysis usage of learnt clause considering LBD

should not alter performances. Once communities have been computed, *sp* starts to operate the CDCL algorithm (as others), and the initial sharing strategy is augmented by clauses characterized by an LBD $= 4$ and a COM ≤ 3. Preliminary experiments showed that *sp* does not need more than a minute to finish Louvain for almost all instances of all benchmarks. Therefore, the augmented filter is activated early in the resolution of a formula.

For the evaluation, we used the main benchmark from SAT competitions 2016[10], 2017[11], 2018[12], and 2019[13]. All solvers were launched on the same machines and with the same configuration than Section 3.2.

The results we observed are discussed in the next section.

5.2 Results and Discussion

Table 2 presents the experimental results on the aforementioned benchmarks. When considering the number of solved instances, we clearly observe that the new sharing strategy outperforms the one used in `P-MCOMSPS`, on all the SAT competition benchmarks.

Coming to the PAR-2 metrics, things seem to be more mitigated. We study the sharing strategy of `P-MCOMSPS` to find an explanation: `P-MCOMSPS` starts the resolution with a low LBD threshold and increments this threshold if it deems

[10] https://baldur.iti.kit.edu/sat-competition-2016/downloads/app16.zip
[11] https://baldur.iti.kit.edu/sat-competition-2017/benchmarks/Main.zip
[12] http://sat2018.forsyte.tuwien.ac.at/benchmarks/Main.zip
[13] http://satcompetition.org/sr2019benchmarks.zip

that the shared clauses throughput is not sufficient. This incremental strategy can help the solver to learn relevant information leading to the resolution of some edge cases. On the contrary, our restrictive sharing strategy can miss those relevant information for these particular cases.

	Solvers	PAR-2	# solved instances
2019	`P-MCOMSPS-L4C3`	386h14	**274**
	`P-MCOMSPS`	**380h03**	273
2018	`P-MCOMSPS-L4C3`	**408h01**	**268**
	`P-MCOMSPS`	430h13	258
2017	`P-MCOMSPS-L4C3`	**352h31**	**238**
	`P-MCOMSPS`	355h42	237
2016	`P-MCOMSPS-L4C3`	355h08	**183**
	`P-MCOMSPS`	**354h39**	182
All together	`P-MCOMSPS-L4C3`	**1 501h54**	**963**
	`P-MCOMSPS`	1 520h37	950

Table 2: Evaluation of the performance of `P-MCOMSPS-L4C3`

Finally, as our strategy is based on the study made on a sequential solver, we want to verify the evolution of the usage ratio in the parallel context. Let us conduct a new evaluation: using the same protocol as the one developed in the sequential setting of Section 4.2, we compute the clause usage ratio in propagation and conflict analysis of our parallel solver `P-MCOMSPS-L4C3`. The resulting number is the sum of all underlying sequential CDCL engines. These logs concern 100 instances (randomly taken) from the benchmark of the SAT competition 2018.

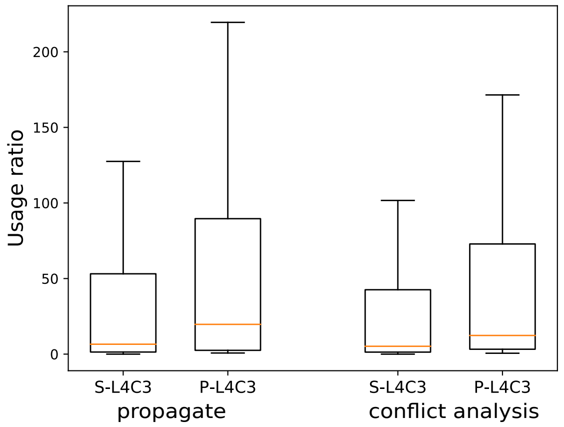


Fig. 6: Efficiency of our proposed sharing strategy.

The collected data are presented in the box-plots of Fig. 6 (the left pair shows propagation and the right pair displays conflict analysis). Box-plots noted S-L4C3 represent the usage ratio of the corresponding set of clauses in `Maple-COMSPS` (the used sequential solver), while those noted P-L4C3 do the same for `P-MCOMSPS-L4C3`.

The shared clauses in `P-MCOMSPS-L4C3`, clearly have a positive impact on the intrinsic behaviour of the underlying sequential engines. They bring new useful information for both unit propagation and conflict analysis procedures. For example, comparing the usage ratio in unit propagation of clauses in S-L4C3 and P-L4C3, we see that the ratio of these clauses goes beyond 50 in only $\approx 25\%$ of the problems for S-L4C3, reaching an upper bound of 130. In P-L4C3, this ratio goes beyond 90 in $\approx 25\%$ of the problems and reaches an upper bound slightly greater than 200. The same observation holds for the conflict analysis procedure. Besides, the medians for box-plots of the parallel approach are all higher than the corresponding medians of the sequential ones.

6 Related Works

The notable community structure of industrial SAT formulas has been identified in [1]. Newsham et al. think that such a structure is one main reason for the noticeable performances of SAT solvers on industrial problems [29].

Thus, several works exploit communities to improve solver performances. It has been used to split formulas to divide the work. Ansótegui et al. developed a pre-processor that solves community-based sub-formulas [2]. The aim is to collect useful information used to solve the whole formula afterwards. Martins et al. show that splitting the formula using communities helps for solving Max-SAT problem in parallel [25]. Community structure has also been used to diversify the decision order of workers in the context of parallel SAT portfolios [31].

It is also worth noting that another concept from the graph theory has been successfully used within the SAT context, namely, the centrality. Katsirelos et al. exhibited that variables selected for branching based on VSIDS are likely to have high eigenvector centrality [18]. The *betweenness centrality* has been incorporated successfully to CDCL: for branching, by using special bonus factors while bumping VSIDS of highly central variables [16]; and for cleaning learnt clauses database, by giving more chance to central clauses (the one with more central variables) [17].

Besides, multiple works present metrics to improve clause sharing for parallel SAT solving. `Penelope` [3] implements the progress saving based quality measure (*psm*). The *psm* of a clause is the size of the intersection between the clause and the phase saving of the solvers. The greater is the psm the more likely the clause will be satisfied. While receiving learnt clauses, a worker can decide to keep them or not. The drawback is that clauses are exchanged and then filtered which can induce some overhead, and an a priori criteria such as LBD is often used as a balance. In `Syrup` parallel solver [6], when a worker learns a clause, it waits for the clause to be used at least once before sending it to the others. The idea behind this is to send only clauses that seem to be useful because already used locally.

While most of the approaches focus on limiting the number of exchanged clauses, Lazaar et al. proposed to select workers allowed to communicate together [20]. This selection is formalized as a multi-armed bandit problem and several metrics are explored as gain functions: size, LBD, activity.

7 Conclusion and Future Works

Most of parallel SAT solvers use local quality metrics (the most relevant being LBD) to select learnt clauses that should be shared. In this paper, we proposed to combine this metric with a more global quality measure (COM) based on the community structure of the input SAT formulas. The guiding principle is to use the community criterion as a filter for set of clauses selected by LBD, in other to increase the usage ratio of shared clauses.

We have designed a tool to track and report learnt clauses' characteristics in a sequential context. As a result of this analysis, we derived a learnt clause sharing policy, which combines LBD and COM, in a parallel context. We attested this strategy by implementing it in `P-MCOMSPS`, which outperforms the competing solver on benchmarks from the SAT competition in 2016, 2017, 2018, and 2019.

We have in mind different ways to improve this work. First, we can look for a more dynamic approach in our filtering method. This would allow to address the problem we mentioned in the analysis of the result in Section 5.2. We noted that some instances benefit from an "unlimited" sharing of clauses: we believe that we could use a delta around some threshold value for the LBD metrics, while maintaining our threshold for the COM value. We could study the shared clauses throughput required for different types of instances as well as the throughput allowed by different types of hardware to increase or decrease the LBD accordingly. It is worth noting that Hamadi et al. developed a similar idea but using the size as a metric [14].

Second, we would like to study the effect of using communities as a metric for garbage collection. This latter is one of the main components of a SAT solver, as the solver can learn millions of clauses while exploring solutions : keeping all these clauses slows down the propagation and leads to memory problems. This is amplified in the parallel context where a sequential solver has its own set of learnt clauses enriched by other solvers too. In `P-MCOMSPS`, clauses are deleted depending on their LBD values. It makes sense to use a local metrics for the logic of a local component. However, the encouraging results shown in this paper let us think that extending the use of communities to garbage collection would improve furthermore the sequential engines.

References

1. Ansótegui, C., Giráldez-Cru, J., Levy, J.: The community structure of sat formulas. In: Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT). pp. 410–423. Springer (2012)

2. Ansótegui, C., Giráldez-Cru, J., Levy, J., Simon, L.: Using community structure to detect relevant learnt clauses. In: Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT). pp. 238–254. Springer (2015)
3. Audemard, G., Hoessen, B., Jabbour, S., Lagniez, J.M., Piette, C.: Revisiting clause exchange in parallel sat solving. In: Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT). pp. 200–213. Springer (2012)
4. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern sat solvers. In: Proceedings of the 21st International Joint Conferences on Artifical Intelligence (IJCAI). pp. 399–404. AAAI Press (2009)
5. Audemard, G., Simon, L.: Glucose in the sat 2014 competition. In: Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions. p. 31. Department of Computer Science, University of Helsinki, Finland (2014)
6. Audemard, G., Simon, L.: Lazy clause exchange policy for parallel sat solvers. In: Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT). pp. 197–205. Springer (2014)
7. Balyo, T., Sinz, C.: Parallel satisfiability. In: Handbook of Parallel Constraint Reasoning, pp. 3–29. Springer (2018)
8. Biere, A.: Splatz, lingeling, plingeling, treengeling, yalsat entering the sat competition 2016. In: Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions. p. 44. Department of Computer Science, University of Helsinki, Finland (2016)
9. Biere, A., Heule, M., van Maaren, H.: Handbook of Satisfiability, vol. 185. IOS press (2009)
10. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment **2008**(10), P10008 (2008)
11. Brandes, U., Delling, D., Gaertler, M., Gorke, R., Hoefer, M., Nikoloski, Z., Wagner, D.: On modularity clustering. IEEE transactions on knowledge and data engineering **20**(2), 172–188 (2007)
12. Cook, S.A.: The complexity of theorem-proving procedures. In: Proceedings of the 3rd ACM Symposium on Theory of Computing (STOC). pp. 151–158. ACM (1971)
13. Eén, N., Biere, A.: Effective preprocessing in sat through variable and clause elimination. In: Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT). pp. 61–75. Springer (2005)
14. Hamadi, Y., Jabbour, S., Sais, J.: Control-based clause sharing in parallel sat solving. In: Autonomous Search, pp. 245–267. Springer (2011)
15. Hamadi, Y., Jabbour, S., Sais, L.: Manysat: a parallel sat solver. Journal on Satisfiability, Boolean Modeling and Computation **6**(4), 245–262 (2009)
16. Jamali, S., Mitchell, D.: Centrality-based improvements to cdcl heuristics. In: Proceedings of the 21st International Conference on Theory and Applications of Satisfiability Testing (SAT). pp. 122–131. Springer (2018)
17. Jamali, S., Mitchell, D.: Simplifying cdcl clause database reduction. In: Proceedings of the 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT). pp. 183–192. Springer (2019)
18. Katsirelos, G., Simon, L.: Eigenvector centrality in industrial sat instances. In: Proceedings of the 18th International Conference of Principles and Practice of Constraint Programming (CP). pp. 348–356. Springer (2012)

19. Kautz, H.A., Selman, B., et al.: Planning as satisfiability. In: Proceedings of the 10th European Conference on Artificial Intelligence (ECAI). vol. 92, pp. 359–363 (1992)
20. Lazaar, N., Hamadi, Y., Jabbour, S., Sebag, M.: Cooperation control in parallel sat solving: a multi-armed bandit approach. Tech. Rep. RR-8070, INRIA (2012)
21. Le Frioux, L., Baarir, S., Sopena, J., Kordon, F.: Painless: a framework for parallel sat solving. In: Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT). pp. 233–250. Springer (2017)
22. Le Frioux, L., Metin, H., Baarir, S., Colange, M., Sopena, J., Kordon, F.: painless-mcomsps and painless-mcomsps-sym. In: Proceedings of SAT Competition 2018: Solver and Benchmark Descriptions. pp. 33–34. Department of Computer Science, University of Helsinki, Finland (2018)
23. Liang, J.H., Oh, C., Ganesh, V., Czarnecki, K., Poupart, P.: Maplecomsps, maplecomsps lrb, maplecomsps chb. In: Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions. p. 52. Department of Computer Science, University of Helsinki, Finland (2016)
24. Marques-Silva, J.P., Sakallah, K.: Grasp: A search algorithm for propositional satisfiability. IEEE Transactions on Computers **48**(5), 506–521 (1999)
25. Martins, R., Manquinho, V., Lynce, I.: Community-based partitioning for maxsat solving. In: Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT). pp. 182–191. Springer (2013)
26. Massacci, F., Marraro, L.: Logical cryptanalysis as a sat problem. Journal of Automated Reasoning **24**(1), 165–203 (2000)
27. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient sat solver. In: Proceedings of the 38th Design Automation Conference (DAC). pp. 530–535. ACM (2001)
28. Newman, M.E., Girvan, M.: Finding and evaluating community structure in networks. Physical Review E **69**(2), 026113 (2004)
29. Newsham, Z., Ganesh, V., Fischmeister, S., Audemard, G., Simon, L.: Impact of community structure on sat solver performance. In: Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT). pp. 252–268. Springer (2014)
30. Pipatsrisawat, K., Darwiche, A.: A lightweight component caching scheme for satisfiability solvers. In: Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT), pp. 294–299. Springer (2007)
31. Sonobe, T., Kondoh, S., Inaba, M.: Community branching for parallel portfolio sat solvers. In: Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT). pp. 188–196. Springer (2014)
32. Zhang, H., Bonacina, M.P., Hsiang, J.: PSATO: a distributed propositional prover and its application to quasigroup problems. Journal of Symbolic Computation **21**(4), 543–560 (1996)