Using Resolution Proofs to Analyse CDCL Solvers

Janne I. Kokkala[1,2(✉)] and Jakob Nordström[2,1(✉)]

[1] Lund University, Lund, Sweden
janne.kokkala@cs.lth.se
[2] University of Copenhagen, Copenhagen, Denmark
jn@di.ku.dk

Abstract. We propose that CDCL SAT solver heuristics such as restarts and clause database management can be analysed by studying the resolution proofs produced by the solvers, and by trimming these proofs to extract the clauses actually used to reach the final conclusion.

We find that for non-adaptive Luby restarts higher frequency makes both untrimmed and trimmed proofs smaller, while adaptive restarts based on literal block distance (LBD) decrease proof size further mainly for untrimmed proofs. This seems to indicate that restarts improve the reasoning power of solvers, but that making restarts adaptive mainly helps to avoid useless work that is not needed to reach the end result.

For clause database management we find that switching off clause erasures often, though not always, leads to smaller untrimmed proofs, but has no significant effect on trimmed proofs. With respect to quality measures for learned clauses, activity in conflict analysis is a fairly good predictor in general for a clause ending up also in the trimmed proof, whereas for the very best clauses the LBD score gives stronger correlation. This gives more rigorous support for the currently popular heuristic of prioritizing clauses with very good LBD scores but sorting the rest of the clauses with respect to activity when deciding which clauses to erase. We remark that for these conclusions, it is crucial to use the actual proof found by the solver rather than the one reconstructed from the *DRAT* proof log.

1 Introduction

Boolean satisfiability (SAT) solving is one of the most striking success stories of computer science, but also one of its most puzzling mysteries. Though modern *conflict-driven clause learning (CDCL)* SAT solvers [29,31][1] are used on an every-day basis to solve real-world instances with hundreds of thousands or even millions of variables, there is still a very poor understanding of how they can perform so well on problems that are, after all, widely conjectured to be exponentially hard in the worst case [14,21].

[1] A similar idea in the context of constraint satisfaction problems (CSPs) was independently developed in [5].

© Springer Nature Switzerland AG 2020
H. Simonis (Ed.): CP 2020, LNCS 12333, pp. 427–444, 2020.
https://doi.org/10.1007/978-3-030-58475-7_25

The most important difference between CDCL and classic DPLL backtrack search [15] is in how conflicts guide the search by generating new learned clauses [29] and informing branching decisions [31], and this accounts for most of the performance gain of CDCL over DPLL [24]. Further improvements have been obtained through careful implementation of the basic CDCL algorithm with highly optimized data structures, as well as through the use of sophisticated heuristics such as *activity* [17] or *literal block distance (LBD)* [2] to identify useful clauses, *phase saving* [35] to guide variable decisions, *adaptive restarts* [3,10] to speed up the search, et cetera.

Unfortunately, our scientific understanding of the performance of these heuristics is still very limited. A natural approach to gain insights would seem to be to collect real-world benchmarks and run experiments with different heuristics to study how they contribute to overall performance. This has been done in [24,27], and there are also in-depth studies focusing specifically on, e.g., variable decisions [11] and restart schemes [12,20], but it has been hard to reach clear-cut conclusions from the diverse set of formulas found in real-world benchmark sets. Another approach has been to run experiments on crafted benchmarks [18,23], where detailed knowledge of the theoretical properties of the formulas makes it possible to draw conclusions about solver performance, but although this can uncover intriguing findings, it is not clear to what extent the conclusions are relevant in a real-world setting.

Our Contributions. In this paper, we investigate whether the proofs generated by SAT solvers can shed light on the effectiveness of solver heuristics. When a CDCL solver decides that a formula is unsatisfiable, it does so, in effect, by deriving a proof of contradiction in the *resolution proof system* [7].[2] Once the solver has terminated, such a proof can be trimmed to keep only the subset of clauses needed to reach this conclusion. We study such untrimmed and trimmed proofs obtained from a selection of benchmarks from the SAT competitions [36] in order to gain insights into solver performance, focusing on restarts and clause database management and how they affect the solver reasoning.

It is well-known that frequent restarts are crucial for the performance of CDCL solvers, but it has remained stubbornly open whether such restarts are just a helpful heuristic or whether they fundamentally increase the theoretical reasoning power. This question cannot be settled by experiments, but we give some empirical evidence that the latter alternative might apply by showing that solvers not only run faster with frequent restarts but also reason more efficiently.

In more detail, we study adaptive restarts as in *Glucose* [3,19] and compare to the non-adaptive Luby restarts in *MiniSat* [17,30], but with different multiplicative constants to get non-adaptive restart frequencies in the full range from the most frequent to least frequent adaptive restarts encountered for our benchmarks. For the non-adaptive policy we find that higher restart frequency correlates with smaller proof size for both untrimmed and trimmed proofs. Adaptive restarts yield smaller untrimmed proofs than all non-adaptive restart frequencies, so the effect of adaptiveness is not only about the frequency but also the

[2] Note, though, that this is not quite true for some pre- and inprocessing techniques.

exact timing of the restarts. The improvements from adaptive restarts are not as clear for the trimmed proofs, however. Our interpretation of this is that more frequent restarts improve the reasoning power of solvers, but that adaptive restarts mainly help to abort useless stages of the search process earlier.

When managing the clauses learned during search, there is a tension between on the one hand keeping as many clauses as possible, since they prune the search space and thus make the reasoning stronger, and on the other hand getting rid of them, since as the number of clauses grows the solver has to spend increasing time on handling them, which makes the reasoning slower. Conventional wisdom dictates that solvers should aggressively minimize memory usage, erasing an ever increasing fraction of learned clauses as the running time increases, but there is little scientific understanding of how this affects the quality of the reasoning performed, or of how to assess which clauses should be kept or thrown away.

When we experiment with switching off erasures completely, so that the solver keeps all learned clauses, we see that this most often leads to smaller untrimmed proofs, but far from always. That is, there exist formulas for which, perhaps somewhat counter-intuitively, clause erasures not only make the solver reason faster, but also better. Even more interestingly, even when the untrimmed proofs get smaller, we do not observe any significant effect on the trimmed proofs. This suggests that the core reasoning needed to decide the formula does not get stronger with more clauses in memory, only that these extra clauses help the solver to "focus" and avoid work that turns out to be useless with hindsight.

Regarding which learned clauses are more or less useful for the solver, it is not obvious how to answer this question, since it is unclear how to measure "usefulness". One approach is to fix a non-adaptive strategy for how many clauses should be removed at clause database reduction, and then decide which clauses to erase based on literal block distance (LBD) score or activity, as in *Glucose* and *MiniSat*, respectively. We find that both untrimmed and trimmed proof size is smaller for LBD-based erasure than activity-based erasure, and that (as a control) both are clearly better than randomly choosing which clauses to erase.

Another approach, following [4,25], is to consider learned clauses in the untrimmed proof "useful" if they remain in the proof after trimming. We find that very good LBD scores strongly correlate with appearing in the trimmed proof, but that clause activity is a better predictor over a wider range of values for which learned clauses survive the trimming process. This provides more rigorous evidence for the empirical claim in [34] that the clause database reduction policy should prioritize top LBD scores but gives more weight to clausal activity for clauses with worse LBD scores, a claim that is also supported by the experiments in [22].

A relevant observation in this context is that the conclusions in the last paragraph above rely on using the actual proof found by the solver. It is also possible to reconstruct a resolution proof from the *DRAT* proof logs used in the SAT competitions by applying *DRAT-trim* [41], but we find that such proofs can look quite different from the ones constructed by the solver during search, and so provide less insight into how the solver actually reasoned.

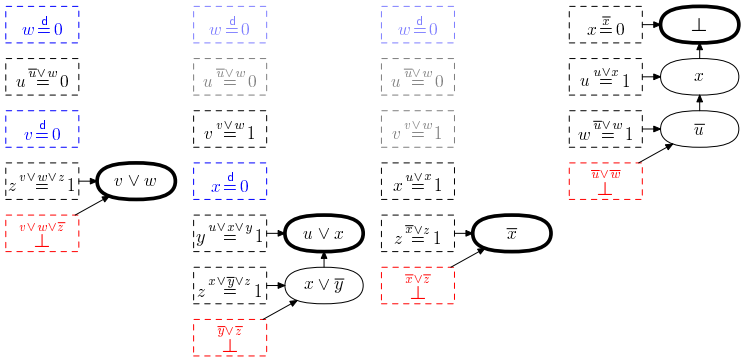
One obvious criticism of this approach is that our notion of usefulness of clauses is narrow—it might well be the case that learned clauses can be helpful for the solver in other ways than by appearing in a final, trimmed proof (as also noted in [4]). Furthermore, even if a clause appears in the trimmed proof, it might be that this appearance comes very soon after the clause was learned, and that after this the clause can safely be thrown away. A more refined approach here is to ask how likely it is at any given point in time that a given clause will be used in the future, a question that was approached in [39] using machine learning techniques. While these are valid points, we nevertheless hope that usage in the trimmed proof can serve as *one* relevant measure providing insights, even though there is certainly room for other measures providing additional information.

Another possible concern is that since we are looking at resolution proofs, we have to limit our attention to only unsatisfiable formulas. Since SAT solvers should work well on both satisfiable and unsatisfiable instances, it could be that we are missing out on important observations by studying only one of these categories of benchmarks. This is also true, but we consider this to be less of a concern. It is in fact possible to come up with a notion of "proofs" also for satisfiable formulas—namely, the learned clauses that guided the solver to the satisfying assignment found, together with all other clauses used to derive these guiding clauses—but we have to leave as interesting future work the task of studying such proofs for satisfiable formulas, and investigating which of our conclusions hold also in this setting and what new observations can be made.

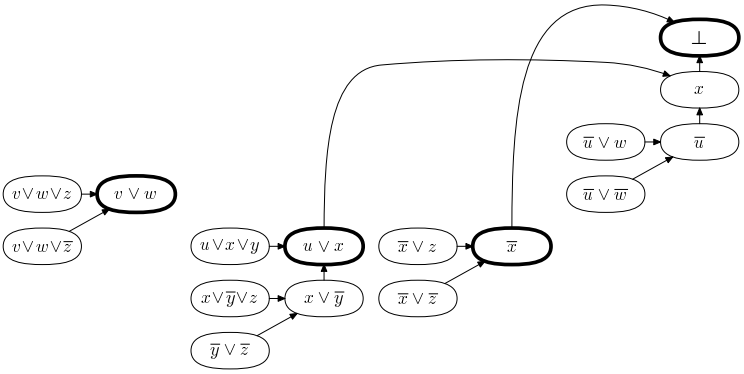
Related Work. A very thorough study of untrimmed and trimmed proofs was performed in [37], where *Glucose* was used to examine the proportion of useful learned clauses across different instances, the effect of shuffling on the number of useful clauses in the input formula, and the correlation of proof size with maximal clause size. Interestingly, usages of older clauses were reported to be more likely to appear in trimmed proofs, but since these experiments were performed only with clause erasures switched on, it was pointed out that this might be due to the solver erasing bad clauses early. *Glucose* was also compared to a solver with *MiniSat*-style policies for restarts and database management, but these experiments did not try to isolate the effects of different heuristics. Furthermore, clause features such as size and LBD score were studied, and LBD was observed to be a better predictor of usefulness than size, but the method used did not allow for an analysis of more dynamic features such as activity.

An analogous idea of trimming appeared in [28], where a dependency graph containing both learned clauses and decided and propagated variables was constructed, and then pruned to contain only the clauses and propagations useful for reaching the final conflict (or the satisfying assignment in the case of satisfiable instances). This was used to study decision heuristics, but the same approach could be harnessed to define a broader notion of clause usefulness, giving credit for useful propagations even if the clause does not appear in conflict analysis.

Outline of This Paper. We start in Sect. 2 by discussing how resolution proofs can be extracted from CDCL solvers. In Sect. 3 we describe our experimental set-up, and in Sect. 4 we present our detailed results. Some final remarks, including suggestions for future research, are made in Sect. 5.



(a) Example CDCL execution (trail grows vertically and shifts right after conflict).



(b) Extracted resolution proof (with long arrows from used learned clauses).

Fig. 1. CDCL execution on formula in (1) and corresponding resolution proof.

2 CDCL SAT Solvers and Resolution Proofs

In this section we discuss briefly, and mostly by example, how resolution proofs can be obtained from CDCL solvers. We refer the reader to, e.g., [13,33] for more details on connections between the theory and practice of SAT solving.

The solver input is a formula in *conjunctive normal form (CNF)* such as

$$\begin{aligned}&(u \vee x \vee y) \wedge (v \vee w \vee z) \wedge (x \vee \overline{y} \vee z) \wedge (v \vee w \vee \overline{z}) \wedge \\ &(\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{u} \vee w) \wedge (\overline{u} \vee \overline{w})\end{aligned} \tag{1}$$

(or, in words, a conjunction of clauses, where every clause is a disjunction of literals, i.e., unnegated or negated variables, with negation denoted by overline). A possible CDCL run for this formula is illustrated in Fig. 1a. At all times, the solver maintains a *trail* of variable assignments, and we show how this trail grows and shrinks vertically as time flows from left to right.

The solver starts by deciding the variable w to false, or 0, which makes the clause $\overline{u} \vee w$ *unit propagate* $u = 0$, since all of this clause except the literal $\overline{u}$

has been falsified. The solver marks the *reason clause* for this propagation on the trail (stacked on the top of the assignment in our illustration). No further propagations can be made, so $w = 0$ and $u = 0$ are all the assignments at *decision level* 1 of the trail. To move things forward the solver has to make a second decision, in our example $v = 0$. Then the clause $v \vee w \vee z$ propagates z to true, or 1, which leads to a *conflict* since the clause $v \vee w \vee \overline{z}$ is now falsified. At the time of conflict, decision level 2 contains $v = 0$ and $z = 1$. During *conflict analysis* the solver *learns* a new clause by applying the *resolution rule*—which resolves two clauses of the form $C \vee z$ and $D \vee \overline{z}$ over the variable z to derive $C \vee D$—to the conflict clause and the reason clauses. In this case, $v \vee w \vee z$ and $v \vee w \vee \overline{z}$ are resolved to yield $v \vee w$, after which the analysis stops (since this is a *unique implication point (UIP)* clause with a single literal from the last decision level).

After learning $v \vee w$, the solver *backjumps* to the *assertion level*, which is the second highest decision level of any literal in the learned clause, by undoing all decisions and propagations at later levels, in our example leaving only $w = u = 0$ at assertion level 1. This causes unit propagation on the learned clause, flipping the value of v (called the *asserting literal*). A new decision $x = 0$ followed by a couple of propagations lead to a second conflict where $u \vee x$ is learned, after which a third conflict results in the learned *unit clause* $\overline{x}$. Unit clauses cause backjumps to decision level 0 (incidentally, this has exactly the same effect as making a *restart*). In our example, this triggers a fourth conflict, and since no decisions have been made the solver can conclude that the formula is unsatisfiable. If, however, we would let the solver run a final conflict analysis, applying the resolution rule to the reasons propagating to the conflict, this would derive the empty clause $\bot$ containing no literals, as shown on the far right in Fig. 1a.

To obtain a resolution proof of unsatisfiability for (1) from Fig. 1a, we start with the final (imagined) derivation of $\bot$, and then go back in time, including the conflict analyses for all clauses used in this derivation, and then the conflict analyses for these clauses, et cetera, leading to the proof visualized in Fig. 1b. During this process, learned clauses that are not needed can be trimmed away. In our example, we see that the first conflict analysis was not needed to decide unsatisfiability. In this way, we obtain untrimmed and trimmed resolution proofs from CDCL executions. Our simplified description ignores aspects like *clause minimization* [40], but such steps also correspond to resolution derivations. Some *preprocessing* steps are not captured by resolution, however, and therefore we analyse CDCL executions on formulas as output by the preprocessor.

In complexity theory, the *proof size* is defined to be the number of clauses in a resolution proof, which in Fig. 1b is 16 for the untrimmed and 13 for the trimmed proof. In this paper, we will be slightly more relaxed and count just the number of learned clauses, so that the untrimmed and trimmed proofs have sizes 4 and 3, respectively. We have verified that this choice does not affect the analysis of our experiments. Conveniently, this means that the size of the untrimmed proof is just the total number of conflicts encountered during execution.

Clause size is defined to be the number of literals in a clause, so that $v \vee w \vee \overline{z}$ has size 3. The *literal block distance (LBD)* of a clause with respect to the current trail is the number of different decision levels represented in the clause. At the time of the first conflict in Fig. 1a, w is assigned at level 1 and v and z at level 2, so the LBD score of $v \vee w \vee \overline{z}$ is 2. The clauses active in the first conflict analysis are $v \vee w \vee \overline{z}$ and $v \vee w \vee z$, and in the second conflict analysis the clauses that take part are $\overline{y} \vee \overline{z}$, $x \vee \overline{y} \vee z$, and $u \vee x \vee y$. Such clauses get their *clause activity* increased by 1, and a mild exponential smoothing is applied to the activity score to give greater weight to the recent history of conflicts.

3 Experimental Set-Up

Let us now describe our CDCL solver configuration and choice of benchmarks.

Solver Configuration We use version 3.0 of *Glucose* [19] (which serves as a basis also for many other modern CDCL solvers), but enhanced to output resolution proofs and to vary restart and clause database management policies.

For ***restarts***, we compare the following policies:

Adaptive restarts. The default in *Glucose*, where, essentially, restarts are triggered when the average LBD score of recently learned clauses becomes bad compared to the overall average.

Luby restarts. As proposed in [20] and used in *MiniSat*, the solver restarts after a predetermined number of conflicts as specified by the *Luby sequence* $1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \ldots$, multiplied by some constant.

Fixed-interval restarts. Restart after a constant number of conflicts.

We study how adaptive restarts affects proof size compared to restarting at predetermined points in time. To investigate whether the effect of adaptivity is mainly to adjust the overall restart frequency or to trigger restarts at specific points in time, we compare to Luby restarts with factors that give similar restart frequencies. Fixed-interval restarts are considered as a theoretically interesting extreme case, though in practice this is too inefficient in terms of running time.

We use Luby sequences with factors 1, 10, 20, 50, 100, and 200. In preliminary experiments with default *Glucose*, for around 95% of unsatisfiable SAT competition benchmarks the total number of restarts are below what would be obtained with Luby restarts with factor 200. We also compare adaptive restarts to the "virtual best Luby solver", picking the best Luby-restarting solver for each benchmark, and the "virtual closest Luby solver" with closest average restart frequency for this particular benchmark. Finally, we run experiments with solvers that restart every 20 conflicts, every 10 conflicts, and every conflict. For all these experiments we use the default *Glucose* clause database management policy.

Concerning ***learned clause deletion***, we investigate how untrimmed and trimmed proof size is affected when the clause database reduction is completely switched off, so that all learned clauses are kept. We run these experiments both for adaptive restarts and for Luby restarts with factor 100 (the *MiniSat* default).

We also consider how the solver chooses which clauses to erase when database reduction is switched on, something we refer to as ***clause assessment***. A first rough description of how the CDCL clause database is managed is as follows. When the solver reaches a certain number of conflicts, a method `reduceDB` is called that sorts the clauses in the database according to some clause assessment criterion, after which the worst half of the clauses are removed (but binary clauses, i.e., clauses of size 2, are never removed). The number of conflicts until the next database reduction is then increased by some constant, meaning that the number of learned clauses in memory after N conflicts will be proportional to $\sqrt{N}$. Glossing over some low-level details (due to space constraints), *Glucose* refines the above model in the following way:

- Clauses with LBD score 1 or 2, so-called *glue clauses*, are never deleted.
- When a clause appears in conflict analysis the LBD score is recomputed, and decreased scores protect from deletion at the next database reduction.
- If many clauses with good LBD scores have been learned, the next clause reduction will be delayed, meaning that more clauses will be kept in memory.

It follows from this that there is a strong feedback loop between the LBD scores and how many learned clauses are kept in memory. As we report in this paper, it is also the case that more clauses in memory tends to yield more efficient reasoning (measured in terms of proof size, not time). If we want to compare different ways of assessing the quality of clauses, we have to break this feedback loop in order to get a fair comparison, since otherwise clause assessment based on LBD might look good just because it leads to more clauses being kept. The problem is, however, that the aggressive clause deletion policy in *Glucose* works well only because the solver keeps more clauses when the LBD scores are good [38].

Therefore, in our clause assessment experiments we use a non-adaptive database reduction strategy that yields clause database sizes that are reasonably close to standard *Glucose*, so that the comparisons will be meaningful, but (almost) never smaller, so that our experiments will not be biased by deleting clauses more aggressively than *Glucose* would do. After some experimentation, the best solution we found was to make each `reduceDB` call erase only 30% of the clauses and to increase the database reduction interval by 4600. This leads to database sizes that are larger than default *Glucose* except for 5% of our benchmarks.

We now have database reduction policy that always erases the same number of clauses regardless of how these clauses are chosen, so that we can study the effect of different clause assessment policies in isolation. Or at least almost: one final problem is that every time a unit clause is learned, all clauses implied by that unit are erased, meaning that the number of clauses in memory shrink, potentially quite significantly. Thus, if a particular clause assessment policy is successful in the sense of leading to more unit clauses being learned, this will make the solver manage memory more aggressively. In contrast to the LBD feedback loop discussed above, we see no way of countering this effect, since not erasing satisfied clauses immediately would also lead to unpredictable effects on the database size (which we cannot explain in detail due to space constraints).

In our clause assessment experiments, we always keep binary clauses and remove the worst 30% of the other clauses sorted according to the following criteria (except for the first default configuration):

LBD+bumps. Default policy in *Glucose* with database size being bumped if the LBD scores of clauses are good enough (evaluated for comparison).
LBD. Simulation of Glucose but with non-adaptive database size policy, prioritizing (a) first glue clauses (LBD score 1 or 2), (b) then clauses with updated LBD, and (c) finally other clauses sorted by LBD (breaking ties by activity).
Activity. Activity in conflicts, with higher activity being better (as in *MiniSat*).
Size. Clause size, with smaller clauses being better.
Random. Random choice of which clauses to erase.

For these experiments we use Luby restarts (with factor 100) to avoid feedback between clause assessment and restarts (except for the default configuration).

Benchmark Selection and Analysis. We ran three separate sets of experiments to measure the effects of restarts, clause deletion, and clause assessment as described above. To select benchmarks for these experiments, we first randomly sampled 200 unsatisfiable instances from the SAT competitions and races 2015–2019 [36] and ran them through the preprocessor of *Glucose* (since the extracted proofs are for CDCL search after preprocessing). Since we want to analyse proofs we cannot deal with time-outs, and so have to select benchmarks solvable by all solver configurations. We therefore ran each solver configuration for all 200 benchmarks, and let each configuration select the 150 instances that were solvest fastest. The final collection of benchmarks for each set of experiments was chosen as the intersection of the sets of benchmarks selected by each solver configuration. Just to give a sense of the computational effort involved, for standard *Glucose* this approach led to running times of around 6,000 s or less. For the restart experiments, we had running times of up to 13,000 s, for experiments with clause deletion switched off up to 55,000 s, and for the clause assessment the control experiment with random erasures resulted in times of up to 173,000 s (2 days). One instance was solved in only 9 conflicts, before any restarts or clause erasures, so it was ignored in the analysis. The number of benchmarks in the final experiments was between 120 and 133.

For each solver configuration and each instance, we collected data about trimmed and untrimmed proof size (where, as mentioned before, the latter is the total number of conflicts during execution), and compared different solver configurations for both trimmed and untrimmed proofs. In order not to give undue weight to the very hardest benchmarks, we consider logarithms of proof sizes. For two different solvers, we use the standard paired t-test to find a 99% confidence interval for the mean of the difference of the logarithms. This confidence interval can be transformed back to a confidence interval for the geometric mean of the ratio of the proof sizes. It is important to note that since we perform multiple experiments and tests, the 99% confidence level cannot be regarded as a proper measure of statistical significance, but the confidence intervals still provide a useful way of understanding the magnitude of the differences.

Features of Useful Clauses. For the experiments with clause deletion switched off, we compare untrimmed and trimmed proofs to see whether different properties of learned clauses can predict whether they will be useful or not, i.e., remain in the final, trimmed, proof. In order to obtain results that could be useful for future solver development, we focus on clause features that the solver could know during execution, rather than on information that can be computed only with hindsight. We consider *static features*, which are determined when the clause is learned, and *dynamic features*, which can change while the solver is running. Since dynamic features can vary over time, what we measure are features of *clause usages* in derivations rather than of the clauses themselves. If a clause in the database is used several times, every usage gives rise to a new data point.

Because features are often used to assess clauses relative to other clauses in the database, and because clauses that are never used by the solver would not appear in our analysis of usages, we also consider the percentile ranks of features in the database at the time of usage. The percentile rank also changes over time when the distribution of features of learnt clauses changes. When collecting data for the percentile ranks, ties are broken randomly.

We collect the following static features computed when the clause is learned:

Size. Size of the learned clause.
Initial LBD. Clause LBD score (with respect to the trail when learned).
Decision level. Decision level of conflict when learned.
Backjump length. Difference of conflict level and assertion level.
Conflicts since restart. Number of conflicts since the latest restart.

We also consider the following dynamic features:

Dynamic LBD. When a clause is learned, its dynamic LBD is set to the initial LBD score. Whenever a clause is used as a reason during conflict analysis, a new candidate LBD score is computed based on the current trail, and the dynamic LBD is updated if the score decreased by at least 2.
Activity. Conceptually speaking, the initial activity of a newly learned clause is 1; it is increased by 1 every time the clause appears in conflict analysis; and all clausal activities are multiplied by a factor $\alpha = 0.999$ after every conflict.

Since solvers will not remove unit or binary clauses, we focus on features for clauses of size at least 3, and the percentile ranks are also computed among these clauses. The reason that clause deletion is switched off is that we do not want the choice of which clauses to erase to bias which clauses seem useful. For the same reason, we use non-adaptive Luby restarts (with factor 100).

By the nature of how CDCL solvers work, we expect some features to correlate strongly with clause usage for trivial reasons. For example, small clauses are more likely to propagate and thus to appear more often in conflict analysis, and will also tend to have low LBD scores. A clause that currently has high activity has been used a lot in conflict analysis, meaning that all other things being equal it is also more likely to show up in the trimmed proof. Such correlations may not say too much about whether the clauses actually contribute to terminating the

search, as they could also have many usages that do not appear in the trimmed proof. To measure the predictive power of a feature, we focus on the *conditional probability* that a usage of a clause appears also in the trimmed proof, conditioning on the value of the feature. For a completely uninformative predictor, this would simply be the ratio of all clause usages in the trimmed proof versus the untrimmed proof. If the conditional probabilities for some values of a feature differ from that, it suggests that the feature can be a predictor for usefulness.

We combine data from multiple benchmarks by summing the absolute counts of usages over all benchmarks. In general, this approach may make a few benchmarks with large proofs dominate. To check whether we have this problem, we performed the same analysis on random subsets of the selected instances. The results were similar, so the analysis appears robust to the effect of single large instances.

4 Results

The results of our experiments, and our analysis of them, are as follows. For full data and source code, see https://doi.org/10.5281/zenodo.3951538.

4.1 Proof Sizes

In Tables 1–3, we present the experimental data for some selected pairwise comparisons of solver configurations. For each pair of solvers and type of proof (untrimmed/trimmed), we calculate the ratio of sizes of the proofs provided by the solvers for each instance. In the tables, we show the quartiles of these ratios in the data and the geometric mean with the 99% confidence interval, computed independently for untrimmed and trimmed proofs.

Comparing different restart frequencies (see Table 1), we find that smaller Luby factors (i.e., faster restarts) tend to give shorter untrimmed and trimmed proofs on average. Restarting at every 20 conflicts gives shorter proofs than Luby restarts with factor 20, but for factor 1 there is no clear difference between Luby and fixed-interval restarts. For fixed-interval restarts every 1, 10, and 20 conflicts, more frequent restarts seem to increase the proof sizes, but the difference is not statistically significant. We interpret this as evidence that up to a certain limit, more frequent restarts generally improve the reasoning power of the solver. Adaptive restarts appear to be better than even the most frequent Luby restart policy, though, yielding clearly smaller untrimmed proofs, and perhaps also slightly smaller trimmed proofs. This seems to indicate that the advantage of adaptive restarts comes mainly from recognizing when the solver is doing useless work and not as much from finding better proofs. One could ask whether adaptive restarts work by simply selecting the best restart frequency for each instance. However, our data gives evidence to the contrary, as choosing the Luby solver with the closest average restart frequency for each instance would perform worse.

Table 1. Comparison of restart policies. Values larger than 1 mean that the first solver generates larger proofs.

		Untrimmed proof				Trimmed proof			
Solvers		quartiles			geom. mean	quartiles			geom. mean
Luby-200	Luby-100	0.93	1.03	1.19	1.06 ± 0.11	0.91	1.01	1.17	1.04 ± 0.08
Luby-100	Luby-50	0.92	1.04	1.19	1.08 ± 0.12	0.95	1.06	1.26	1.11 ± 0.09
Luby-50	Luby-20	0.89	0.99	1.17	0.97 ± 0.11	0.92	1.01	1.12	1.02 ± 0.11
Luby-20	Luby-10	0.95	1.03	1.19	1.07 ± 0.09	0.95	1.04	1.19	1.06 ± 0.10
Luby-10	Luby-1	0.89	1.01	1.16	1.03 ± 0.09	0.94	1.03	1.15	1.06 ± 0.06
Luby-20	Fixed-20	0.78	1.13	1.56	1.13 ± 0.17	0.92	1.18	1.61	1.23 ± 0.15
Luby-1	Fixed-1	0.72	0.96	1.19	0.93 ± 0.12	0.91	1.10	1.28	1.03 ± 0.11
Fixed-20	Fixed-10	0.88	1.02	1.14	0.96 ± 0.09	0.90	1.01	1.10	0.98 ± 0.07
Fixed-10	Fixed-1	0.81	0.98	1.13	0.95 ± 0.10	0.86	0.99	1.09	0.96 ± 0.09
adaptive	Luby-20	0.68	0.84	0.97	0.79 ± 0.09	0.80	0.95	1.07	0.92 ± 0.09
adaptive	Luby Closest	0.63	0.83	1.02	0.75 ± 0.08	0.76	0.91	1.04	0.83 ± 0.09
adaptive	Luby VBS	0.89	1.06	1.24	1.10 ± 0.13	0.99	1.13	1.33	1.19 ± 0.11
adaptive	Fixed-20	0.57	0.90	1.35	0.89 ± 0.14	0.85	1.06	1.58	1.13 ± 0.14

Table 2. Effect of turning clause erasures off with adaptive restarts and Luby-100 restarts. Values larger than 1 mean that the first solver generates larger proofs.

		Untrimmed proof				Trimmed proof			
Solver erasures/restarts		quartiles			geom. mean	quartiles			geom. mean
Off/adaptive	On/adaptive	0.70	0.81	0.96	0.77 ± 0.07	0.86	1.00	1.15	0.98 ± 0.06
Off/Luby	On/Luby	0.67	0.80	0.99	0.77 ± 0.07	0.82	0.98	1.18	0.98 ± 0.08

Turning clause erasures completely off decreases the untrimmed proof size in 83% of the instances, and by 23% on average, but there appears to be no measurable average difference for the trimmed proofs. The results are similar when using Luby restarts with factor 100 instead of adaptive restarts (see Table 2).

Comparing the two popular policies LBD and activity for clause assessment, we see (as shown in Table 3) that using LBD gives significantly smaller proofs. We also find that LBD is better than clause size, which is evidence that LBD contains more useful information than just size despite being strongly correlated with it. Clause size, in turn, seems to be slightly better than activity, but the difference is not statistically significant. Choosing which clauses to erase at random is clearly worse than all other policies, but even so it is not a completely hopeless approach, as it yields shorter proofs than LBD for 10–15% of the instances.[3]

To verify that our solver with fixed database size updates and LBD clause assessment is reasonably close to the actual behaviour of *Glucose* with adaptive

[3] For one of the selected benchmarks the solver with random clause erasures produced a proof too large to analyse with our tool chain, so this data point is missing. However, it would not make any significant difference.

Table 3. Comparison of clause assessment policies. Values larger than 1 mean that the first solver generates larger proofs.

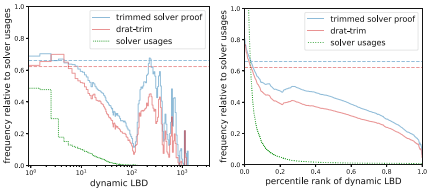
		Untrimmed proof				Trimmed proof			
Solvers		quartiles			geom. mean	quartiles			geom. mean
LBD	activity	0.74	0.92	1.02	0.84 ± 0.09	0.74	0.88	1.00	0.84 ± 0.09
LBD	size	0.79	0.94	1.01	0.91 ± 0.07	0.78	0.91	1.00	0.87 ± 0.07
LBD	random	0.66	0.80	0.97	0.73 ± 0.07	0.61	0.74	0.88	0.66 ± 0.08
size	activity	0.83	1.01	1.23	0.93 ± 0.10	0.85	1.00	1.14	0.96 ± 0.09
activity	random	0.81	0.91	1.00	0.86 ± 0.08	0.73	0.86	0.97	0.79 ± 0.09
LBD+bumps	LBD	1.00	1.17	1.37	1.21 ± 0.12	0.94	1.08	1.23	1.10 ± 0.10
Glucose	LBD	0.83	1.00	1.20	0.97 ± 0.12	0.81	1.00	1.13	0.95 ± 0.12

database size and adaptive restarts, we also compare the proof sizes for these two solvers. There is no statistically significant difference for the proof sizes, and 50% of proof sizes obtained from *Glucose* are within 19% of our LBD model, so we believe that this clause assessment experiment is relevant in practice.

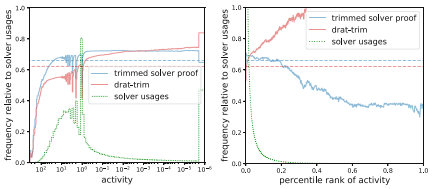
Clause Features. We estimate the conditional probability that a clause usage in the untrimmed proof appears also in the trimmed proof by dividing the sampled frequency distribution of a feature in the trimmed proofs by the frequency distribution in the untrimmed proofs. In Fig. 2, we visualize the computed conditional probabilities for some features. In addition, the plots contain a dashed line that shows the ratio of all clause usages in the trimmed proof versus the untrimmed proof, which is what the graph for an uninformative, completely uncorrelated, predictor would look like. Similar values can also be computed for the *DRAT-trim* proof instead of the trimmed solver proof (although they cannot be interpreted as conditional probabilities since *DRAT-trim* usages are not a subset of solver usages); these are shown in the same plots for comparison. To indicate which values are relevant, plots also show the frequency distribution of all solver usages, transformed for the logarithmic x-axis so that area under the curve corresponds roughly to a probability measure (but with arbitrary scaling).

For dynamic LBD, glue clauses (with LBD scores at most 2) occur in the trimmed proofs more than average, and the top 5% of clauses have clearly larger probability of appearing in the trimmed proof than the rest. In the plot, there is also a peak around the value 250; however, as the solver usage distribution line shows there are not many usages with these values, so this is likely to be an effect of small sample size. Initial LBD and size are somewhat similar, but dynamic LBD is a better predictor for the top clauses than either of them.

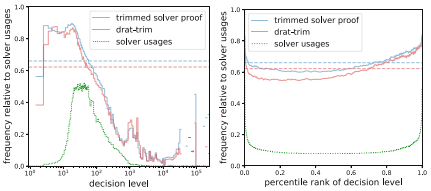
Clauses with very small activity score are sometimes used by the solver, but they tend to be less common in trimmed proofs. Higher values indicate usefulness, except that clauses with very high activity scores (above 30) tend not to be useful; it appears that the solver uses some clauses a lot that are not needed in the final proof. Curiously, activity scores just below small integer values



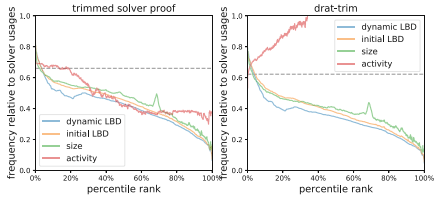
(a) Dynamic LBD.



(b) Activity. Note the reversed x-axis in the absolute value plot.



(c) Decision level when clause was learned.



(d) Comparison of usefulness as a function of percentile rank.

Fig. 2. Sampled conditional probabilities that usages of clauses in the untrimmed proof appears also in the trimmed solver proof, and the analogous ratio for the *DRAT-trim* proof.

are less common in trimmed proofs. These come mostly from clauses that have been used recently and where the activity has not had much time to decay. One possible explanation is that being used many times in short succession may not indicate usefulness, but clauses that are used many times throughout a longer time interval are better.

A comparison of the computed conditional probabilities for percentile ranks of dynamic LBD, initial LBD, size, and activity is shown in Fig. 2d. When comparing the predictive power of the most popular measures, i.e., dynamic LBD and activity, it seems that LBD is a good predictor for the very best clauses, but that activity is relevant for a wider range of values. If we would use the *DRAT-trim* frequency distribution instead, we would not see as clear a difference between dynamic LBD and initial LBD or size. Also, it is clear that low-activity clauses are used by *DRAT-trim* much more often than by the solver.

Measuring the time elapsed from the most recent restart to when a clause is learned does not seem to provide any predictive power. Clauses that cause a backjump of only one decision level seem to appear often in conflict analysis, but tend to be less useful than clauses yielding longer backjumps. The data for the decision level at which a clause is learned seem contradictory: usages with small absolute value are more likely to appear in the trimmed proof, but so are usages with high percentile rank values. Such behaviour could potentially be caused by the distribution of the feature in the database changing over time, but understanding this in detail will require further research.

5 Concluding Remarks

The main philosophy underlying this paper is that in order to gain a better understanding of how CDCL SAT solvers work, it is fruitful to investigate the reasoning that they perform. Since CDCL solvers are search algorithms for resolution proofs when run on unsatisfiable formulas, we can study what kind of proofs they find, and what parts of these proofs are essential for establishing that the formulas are indeed unsatisfiable.

Using this method of analysis, we find that more frequent Luby-style restarts help solvers to produce shorter proofs (even if all too frequent restarts cause too much of a penalty in running time). Making restarts adaptive can significantly decrease proof size further, but mainly for the untrimmed proofs containing all derivations rather than for the trimmed proofs containing only essential clauses. This indicates that adaptive restarts are often successful in helping the solver avoid unnecessary work. When assessing whether a learned clause is likely to be useful, as measured by the probability of the clause appearing in the final, trimmed proof, we find that very good literal block distance (LBD) score is a strong predictor, but that clausal activity appears to be more relevant over a larger range of values. This supports the currently popular approach of prioritizing clauses with low LBD scores but sorting other clauses with respect to activity [34].

We consider our paper, and previous works in a similar spirit such as [28,37], to be only first steps, and see ample scope for future research in this direction. In particular, it would be very interesting to extend our method to satisfiable formulas, by looking at the "proofs" obtained by concatenating the conflict analyses for the learned clauses guiding the solver to the satisfying assignment.

In addition to the heuristics for restarts and clause database management studied in this work, it would be relevant to investigate variable decision heuristics such as VSIDS and phase saving, building on and extending [28]. An arguably even more urgent task is to gain a better understanding of relatively new techniques such as *learned clause minimization* [26] and *chronological backtracking* [32], which have played an important role in the SAT competitions [36] in recent years.

Our data analysis is relatively simple, and there should be room for using more advanced tools. A tempting idea is to combine our approach with the machine learning techniques in [39] (but, importantly, applied on the actual proof found by the solver rather than the one reconstructed by *DRAT-trim*). Also, it would be interesting to study more properties of proofs such as space complexity, and whether theoretical time-space trade-offs as in [1,6,8,9] could show up also in practice, in view of the aggressive memory management in modern solvers.

Finally, it is intriguing that some of our results are quite different from those in [18]. As an example, that paper found that activity-based clause assessment when choosing which clauses to erase is almost equally bad as random, whereas in our work it is clearly better. A natural question is how much of this discrepancy might be due to that we use "applied" SAT competition benchmarks, whereas only crafted, combinatorial formulas were considered in [18].

Acknowledgements. We wish to thank Jo Devriendt and Stephan Gocht for many interesting discussions, and helpful suggestions, throughout the project. We are also grateful to the anonymous reviewers, who helped improve the exposition considerably.

The computational experiments used resources provided by the Swedish National Infrastructure for Computing (SNIC) at the High Performance Computing Center North (HPC2N) at Umeå University. The authors were supported by the Swedish Research Council (VR) grant 2016-00782, and the second author also received funding from the Independent Research Fund Denmark (DFF) grant 9040-00389B.

References

1. Alwen, J., de Rezende, S.F., Nordström, J., Vinyals, M.: Cumulative space in black-white pebbling and resolution. In: Proceedings of the 8th Innovations in Theoretical Computer Science Conference (ITCS 2017). Leibniz International Proceedings in Informatics (LIPIcs), vol. 67, pp. 38:1–38:21, January 2017
2. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009), pp. 399–404, July 2009
3. Audemard, G., Simon, L.: Refining restarts strategies for SAT and UNSAT. In: Milano, M. (ed.) CP 2012. LNCS, pp. 118–126. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33558-7_11
4. Audemard, G., Simon, L.: Lazy clause exchange policy for parallel SAT solvers. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 197–205. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09284-3_15
5. Bayardo Jr., R.J., Schrag, R.: Using CSP look-back techniques to solve real-world SAT instances. In: Proceedings of the 14th National Conference on Artificial Intelligence (AAAI 1997), pp. 203–208, July 1997
6. Beame, P., Beck, C., Impagliazzo, R.: Time-space tradeoffs in resolution: superpolynomial lower bounds for superlinear space. SIAM J. Comput. **45**(4), 1612–1645 (2016). Preliminary version in STOC '12
7. Beame, P., Kautz, H., Sabharwal, A.: Towards understanding and harnessing the potential of clause learning. J. Artif. Intell. Res. **22**, 319–351 (2004). Preliminary version in IJCAI '03
8. Beck, C., Nordström, J., Tang, B.: Some trade-off results for polynomial calculus. In: Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC 2013), pp. 813–822, May 2013
9. Ben-Sasson, E., Nordström, J.: Understanding space in proof complexity: separations and trade-offs via substitutions. In: Proceedings of the 2nd Symposium on Innovations in Computer Science (ICS 2011), pp. 401–416, January 2011
10. Biere, A.: Adaptive restart strategies for conflict driven SAT solvers. In: Kleine Büning, H., Zhao, X. (eds.) SAT 2008. LNCS, vol. 4996, pp. 28–33. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79719-7_4
11. Biere, A., Fröhlich, A.: Evaluating CDCL variable scoring schemes. In: Heule, M., Weaver, S. (eds.) SAT 2015. LNCS, vol. 9340, pp. 405–422. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24318-4_29
12. Biere, A., Fröhlich, A.: Evaluating CDCL restart schemes. In: Proceedings of Pragmatics of SAT 2015 and 2018. EPiC Series in Computing, vol. 59, pp. 1–17, March 2019. https://easychair.org/publications/paper/RdBL

13. Buss, S., Nordström, J.: Proof complexity and SAT solving (2020). Chapter to appear in the 2nd edition of Handbook of Satisfiability. Draft version available at http://www.csc.kth.se/~jakobn/research/
14. Calabro, C., Impagliazzo, R., Paturi, R.: The complexity of satisfiability of small depth circuits. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 75–85. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-11269-0_6
15. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. Commun. ACM **5**(7), 394–397 (1962)
16. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24605-3_37
17. Eén, N., Sörensson, N.: An extensible SAT-solver [extended version 1.2] (2004). http://minisat.se/downloads/MiniSat.pdf. Updated version of [16]
18. Elffers, J., Giráldez-Cru, J., Gocht, S., Nordström, J., Simon, L.: Seeking practical CDCL insights from theoretical SAT benchmarks. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018), pp. 1300–1308, July 2018
19. The Glucose SAT solver. http://www.labri.fr/perso/lsimon/glucose/
20. Huang, J.: The effect of restarts on the efficiency of clause learning. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007), pp. 2318–2323, January 2007
21. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. J. Comput. Syst. Sci. **62**(2), 367–375 (2001). Preliminary version in CCC '99
22. Jamali, S., Mitchell, D.: Centrality-based improvements to CDCL heuristics. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) SAT 2018. LNCS, vol. 10929, pp. 122–131. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94144-8_8
23. Järvisalo, M., Matsliah, A., Nordström, J., Živný, S.: Relating proof complexity measures and practical hardness of SAT. In: Milano, M. (ed.) CP 2012. LNCS, pp. 316–331. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33558-7_25
24. Katebi, H., Sakallah, K.A., Marques-Silva, J.P.: Empirical study of the anatomy of modern SAT solvers. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 343–356. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21581-0_27
25. Katsirelos, G., Sabharwal, A., Samulowitz, H., Simon, L.: Resolution and parallelizability: barriers to the efficient parallelization of SAT solvers. In: Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI 2013), pp. 481–488, July 2013
26. Luo, M., Li, C.M., Xiao, F., Manyá, F., Lü, Z.: An effective learnt clause minimization approach for CDCL SAT solvers. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017), pp. 703–711, August 2017
27. Lynce, I., Marques-Silva, J.P.: Building state-of-the-art SAT solvers. In: Proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002), pp. 166–170. IOS Press, May 2002
28. Malik, S., Ying, V.A.: On the efficiency of the VSIDS decision heuristic, August 2016. presentation at the workshop Theoretical Foundations of SAT Solving. Slides http://www.fields.utoronto.ca/sites/default/files/talk-attachments/SharadMalik-Fields2016.pdf
29. Marques-Silva, J.P., Sakallah, K.A.: GRASP: a search algorithm for propositional satisfiability. IEEE Trans. Comput. **48**(5), 506–521 (1999). Preliminary version in ICCAD '96

30. The MiniSat page. http://minisat.se/
31. 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 2001), pp. 530–535, June 2001
32. Nadel, A., Ryvchin, V.: Chronological backtracking. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) SAT 2018. LNCS, vol. 10929, pp. 111–121. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94144-8_7
33. Nordström, J.: On the interplay between proof complexity and SAT solving. ACM SIGLOG News **2**(3), 19–44 (2015)
34. Oh, C.: Improving SAT solvers by exploiting empirical characteristics of CDCL. Ph.D. thesis, New York University (2016). https://cs.nyu.edu/media/publications/oh_chanseok.pdf
35. Pipatsrisawat, K., Darwiche, A.: A lightweight component caching scheme for satisfiability solvers. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 294–299. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72788-0_28
36. The international SAT Competitions web page. http://www.satcompetition.org
37. Simon, L.: Post mortem analysis of SAT solver proofs. In: Proceedings of the 5th Pragmatics of SAT Workshop. EPiC Series in Computing, vol. 27, pp. 26–40, July 2014. https://easychair.org/publications/paper/N3GD
38. Simon, L.: Personal communication (2018)
39. Soos, M., Kulkarni, R., Meel, K.S.: CrystalBall: gazing in the black box of SAT solving. In: Janota, M., Lynce, I. (eds.) SAT 2019. LNCS, vol. 11628, pp. 371–387. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24258-9_26
40. Sörensson, N., Biere, A.: Minimizing learned clauses. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 237–243. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02777-2_23
41. Wetzler, N., Heule, M.J.H., Hunt, W.A.: DRAT-trim: efficient checking and trimming using expressive clausal Proofs. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 422–429. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09284-3_31