


Learn to Unlearn

Bernhard Gstrein ✉ 

University of Freiburg, Germany

Florian Pollitt ✉ 

University of Freiburg, Germany

André Schidler ✉ 

University of Freiburg, Germany

Mathias Fleury ✉ 

University of Freiburg, Germany

Armin Biere ✉ 

University of Freiburg, Germany

Abstract

Clause learning is a significant milestone in the development of SAT solving. However, keeping all learned clauses without discrimination gradually slows down the solver. Thus, selectively removing some learned clauses during routine database reduction is essential. In this paper, we reexamine and test several long-standing ideas for clause removal in the modern solver Kissat. Our experiments show that retaining all clauses alters performance in all instances. For satisfiable instances, periodically removing all learned clauses surprisingly yields near state-of-the-art performance. For unsatisfiable instances, it is vital to always keep some learned clauses. Building on the influential Glucose paper, we find that it is crucial to always retain the clauses most likely to help, regardless of whether they are ranked by size or LBD in practice. Another key factor is whether a clause was used recently during conflict resolution steps. Eagerly keeping used clauses improves all unlearning strategies.

2012 ACM Subject Classification Theory of Computation → Automated Reasoning

Keywords and phrases Satisfiability solving, learned clause recycling, LBD

Digital Object Identifier 10.4230/LIPIcs.SAT.2025.26

Supplementary Material *Software (Source Code)*: <https://github.com/texmex76/kissat-cr> [16]

Dataset (Log Files): <https://doi.org/10.5281/zenodo.15146408> [17]

Acknowledgements This work was supported by the state of Baden-Württemberg through bwHPC, the German Research Foundation (DFG) grant INST 35/1597-1 FUGG, and a gift from Intel Corp.

1 Introduction

Modern SAT solvers following the conflict-driven clause learning (CDCL) paradigm [21] or related algorithms [28], reduce the search space by learning clauses that boost propagation power. Thanks to solver efficiency and hardware advances, they can now learn many more clauses and solve much larger instances. However, these clauses (also called “no-goods” [28]) gradually slow down the solver because the efficiency of most algorithms depends on the size of their data structures. Furthermore, retaining every clause turns CDCL into an algorithm requiring exponential space, similar to reverting from DPLL [6] to DP [7]. Thus, reducing the number of clauses by *unlearning* is crucial for good performance. In this paper, we revisit established clause unlearning strategies and evaluate their impact on our state-of-the-art SAT solver KISSAT [4], winner of all tracks of the SAT Competition 2024.

Several research efforts have tried to mitigate this problem. Already in 1993, in parallel to the seminal work on CDCL [24], Ginsberg [12] presented a polynomial space variant of conflict-driven no-good learning, which he later adapted in 2015 to allow arbitrary restarts [13] without



© Bernhard Gstrein, Florian Pollitt, André Schidler, Mathias Fleury, and Armin Biere;
licensed under Creative Commons License CC-BY 4.0

28th International Conference on Theory and Applications of Satisfiability Testing (SAT 2025).

Editors: Jeremias Berg and Jakob Nordström; Article No. 26; pp. 26:1–26:12

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

losing completeness. However, state-of-the-art solvers have not adopted this approach.

On the practical side, various unlearning schemes have been proposed: keeping clauses up to a fixed size [8], using relevance-based criteria [20], combining different approaches [14, 18, 24] and more. In our experiments, we revisit size-based unlearning in particular. More recently, CDCL solvers have started to predict the *future* usefulness of a clause by measuring either how often it is used (its activity) [9, 14] or how recently it has been used (its age) [11, 14, 15] during conflict analysis. This information is then used to rank clauses for retention or deletion. The seminal GLUCOSE [1] introduced the new ranking scheme *literal block distance* (LBD) and, more importantly, the idea of keeping specific (glue) clauses unconditionally.

We highlight the importance of clause unlearning in our first experiment (Section 3), where we evaluate KISSAT without any unlearning (a configuration previously studied by Chanseok Oh [23]) as well as periodically unlearning all learned clauses. The experiment shows that keeping all clauses degrades performance on all instance types. Interestingly, while periodic unlearning of all clauses yields nearly state-of-the-art performance on satisfiable instances, it significantly harms performance on unsatisfiable ones.

Our second experiment (Section 4) revisits the favored approach of MINISAT [9], where clause *activity* is used to decide which clauses to unlearn. In this scheme, a clause’s activity increases each time it is used to derive a new clause, while unused clauses are penalized. In our experiments, the performance of KISSAT with activity-based unlearning is unsatisfactory.

In 2009, GLUCOSE [1] introduced a new usefulness criterion based on the propagation power of clauses, approximated by LBD (the number of different decision levels in a clause when it is learned or propagates). While MINISAT ranks clauses based on activity and unlearns a fixed percentage, GLUCOSE sorts clauses into two tiers and unconditionally keeps the lower-tier (critical) clauses. Our experiments in Section 5 confirm that retaining only critical clauses is more important than the difference between using clause size or LBD as the ranking metric. Similar findings regarding the indifference between ranking schemes have also been observed in a related experiment [10] focusing on combinatorial benchmarks.

In 2019, a machine-learning-based approach was used to analyze clause usefulness, which we revisit in Section 6. In that work, the usefulness of a clause was estimated by generating proofs, reconstructing dependencies, and classifying core clauses as useful. Soos et al. [27] identified, that clauses which have been *used* to derive a new clause since the last unlearning round should be kept. This effectively reduces clause activity to a single binary indicator. While this metric alone performs poorly, combining it with other criteria consistently improves solver performance. In fact, such a combination is one of our best configurations and achieves performance similar to the base version of KISSAT while being much easier to implement.

Finally, in Section 7, we discuss the state-of-the-art and KISSAT. Since KISSAT uses LBD as a key metric for estimating a clause’s usefulness, we analyze the actual distribution of LBDs in used clauses. While the distribution usually shows peaks that match our definition of critical clauses, some instances have a minimal LBD far above our critical threshold. We address this by proposing a new dynamic tier system with adaptive thresholds to identify critical clauses, though this scheme does not improve performance.

2 Background and Experimental Setup

We refer the reader to the corresponding chapter of the *Handbook of satisfiability* [21] for an introduction to conflict-driven clause learning (CDCL), i.e., CDCL applies unit propagation after decisions, finds conflicts and then learns new clauses to change the focus of the search, interleaved with restarts, inprocessing [19], as well as unlearning, the focus of this paper.

```

unlearn (some sub-set of the learned clauses  $F$ )
1  backtrack to decision level zero (root-level)           // make sure all clauses can be deleted
2  optionally update critical tier limits dynamically      // Sect. 7
3  initialize  $G$  as an empty list of candidate clauses to unlearn
4  for all learned clauses  $C \in F$ 
5      check the used flag of  $C$ , i.e., whether  $C$  was used during learning since the last unlearn
6      if used continue but reset used flag of  $C$  in any case // Sect. 6
7      if  $C$  is considered critical continue                // Sect. 5 (2-tier system)
8      otherwise add  $C$  to candidate set  $G$ 
9  sort  $G$  by “predicted usefulness” ranking function      // Sect. 4 (activity), Sect. 5 (LBD)
10 remove from  $F$  the least useful fraction  $f \cdot |G|$  of clauses in  $G$  // Sect. 3
    // where  $f$  is the target fraction to remove, e.g.,  $f = 50\%$  removes half of  $G$ ,
    //  $f = 0\%$  no clause (no-unlearn) and  $f = 100\%$  all remaining candidates (unlearn-all)
11 Schedule next unlearning

```

■ **Algorithm 1** Unlearning is part of the *unlearn* function, scheduled regularly within the CDCL loop after a certain number of conflicts occurred and clauses have been learned. We want to understand which clauses in *unlearn* should be thrown-away, i.e., “unlearned”, and which should be kept. More aspects on the KISSAT implementation of *unlearn* (in `reduce.c`) are discussed in appendix A.

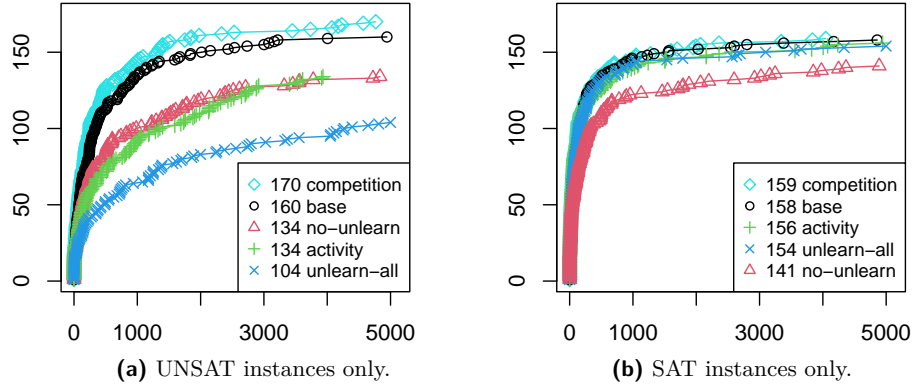
In Alg. 1 we show pseudo-code for the implementation of *unlearn*, which is called periodically from the CDCL loop according to some unlearning schedule. Clauses are only removed during unlearning, so even *unlearn-all* keeps clauses for some time: it only throws away all learned clauses in each unlearning round. In this paper, we vary different parts of this Alg. 1. Changes and their effects are outlined in the respective sections below.

As initial starting point of our investigations we used the **competition** version of the award-winning SAT solver KISSAT [4] from the SAT Competition 2024. It won all main-tracks of this most recent competition and thus is considered the state-of-the-art. In order to reduce influence on and by other heuristics while varying aspects of unlearning, we have simplified the competition version in two ways. First, we let KISSAT restart before unlearning clauses, as otherwise, the solver is forced to keep clauses which are used as reasons for assigned variables. Second, we fix the fraction of removed clauses to be the same for the entire run of the solver, specifically 75%. In the **competition** version it slowly increases from 60% to 90%. The resulting **base** version is the actual starting point for the other variants considered.

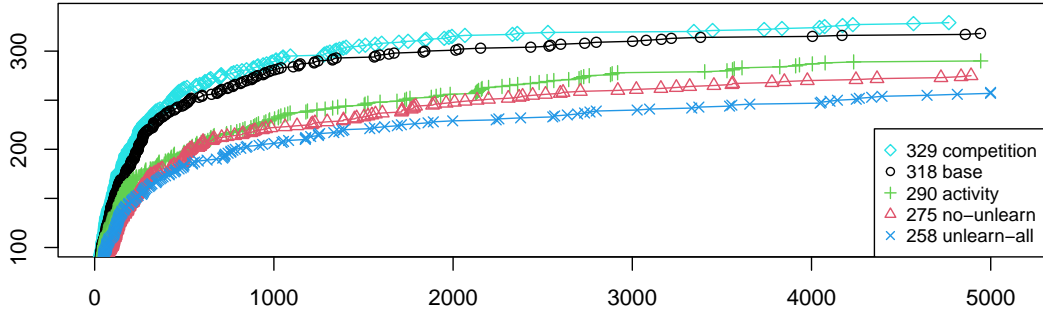
Throughout this paper, all experiments are conducted with the 400 problems from the SAT Competition 2024, following *The SAT Practitioner’s Manifesto* [5] and solved on the **bwForCluster** Helix with AMD Milan EPYC 7513 CPUs, a memory limit of 16 GB per solver run and a time limit of 5000 s. We consistently present results as cumulative distribution functions (CDFs), where the x -axis gives the time and the y -axis shows the number of solved problems (400 total for the entire benchmark set or 200 total when only UNSAT or SAT).

3 Importance of Unlearning

We first evaluate the configurations *unlearn-all* and *no-unlearn* which periodically unlearn all and never unlearn any clauses, respectively. We observe an interesting effect: unlearning all clauses is only very modestly harmful on satisfiable instances (Fig. 2b). It appears sufficient to guess a model to solve those instances and learned clauses only seem to be instrumental to locally guide the solver towards this model, and should be unlearned fast. However,



■ **Figure 2** Comparing unlearning strategies: full competition; simplified base; no-unlearn (retain all clauses); unlearn-all (periodically unlearn all clauses); activity-based (*cf.* Sect. 4).



■ **Figure 3** Comparing the same unlearning strategies as in Fig. 2 but on all competition instances.

116 unlearn-all is detrimental to solving unsatisfiable instances (Fig. 2a).

117 Keeping all clauses (**no-unlearn**) reduces KISSAT's performance for both classes, but is
 118 not as harmful on unsatisfiable instances as expected. It further has a negative impact on
 119 memory usage of course as for **no-unlearn** accumulated memory usage goes from 167 GB to
 120 240 GB. On the other hand, the number of conflicts on the 96 shared solved unsatisfiable
 121 instances of the **no-unlearn** configuration is reduced to 0.7 the number of conflicts in the
 122 **base** configuration and goes up for **unlearn-all** by more than factor of 10. Chanseok Oh [23]
 123 reached similar conclusions for **unlearn-all**.

124 In general, we observe that configurations retaining more clauses perform better on
 125 unsatisfiable instances and worse on satisfiable instances, while unlearning more clauses tends
 126 to improve performance on satisfiable instances but hinders solving unsatisfiable instances.
 127 This might be exploited if the expected result for a particular instance is known.

128 4 Activity as Usefulness Prediction

129 The solver MINISAT [9] implements a common strategy for clause retention: A clause's
 130 *activity*—reflecting how often it contributes to conflict analysis—determines its usefulness.
 131 Each time a clause participates in deriving a learned clause, its activity increases; new
 132 clauses get the current global clause activity increment. The global clause activity increments
 133 grows exponentially with each conflict (penalizing unused clauses), and when values become
 134 too large, scores are rescaled. This is akin to the (E)VSIDS heuristic [9, 22] for variables.
 135 Although alternative metrics like propagation counts have been explored, activity-based

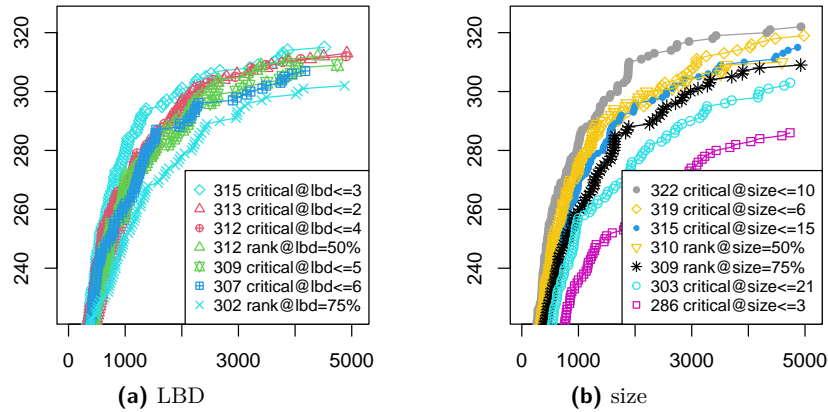


Figure 4 Comparing critical thresholds with rank (using two target fractions f for lbd and size).

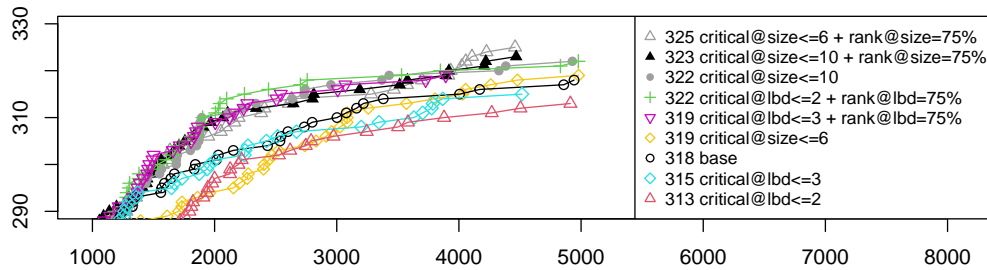


Figure 5 Effect of combining critical and rank with the best configurations from Fig. 4.

heuristics perform best. This aligns with Simon’s observation [25] that roughly 90% of propagations are unused, indicating resolved clauses in conflict analysis drive the search.

We implemented an activity branch on top of the `base` branch (one of 42 for this study [16]) to test how well activity predicts clause usefulness in KISSAT. It keeps the aggressive scheduling used in all experiments but modifies the `unlearn` function (see Alg. 1): the target fraction is set to 50% as in MINISAT, but the `used` flag and critical metrics are ignored.

Figure 3 shows that activity-based unlearning performs worse than our baseline. On satisfiable instances (Fig. 2b) its performance mirrors `unlearn-all`, on unsatisfiable instances (Fig. 2a) it resembles `no-unlearn`. Since satisfiable instances gain little from learned clauses, activity-based unlearning gives mixed results and is a weak predictor by modern standards.

5 Critical Clauses and LBD as Usefulness Prediction

In 2009, Audemard and Simon [1] introduced two key modifications to MINISAT, resulting in GLUCOSE. These include the LBD score and a policy of always retaining high-quality ($\text{LBD} \leq 2$) clauses—termed *glue clauses* by the authors, but called *critical* here to avoid confusion. These changes enabled more aggressive restarts and unlearning by keeping fewer clauses. This yields a *2-tier system*: top-ranked clauses (tier 1) are always retained; others (tier 2) are ranked and selectively unlearned.

The term “LBD” has two definitions. In [1], it counts the number of decision levels in a learned clause before backjumping. Later work [2, 3] updates LBD upon propagation, based on levels after backjumping (one less than the original). To reconcile this, GLUCOSE adds 1 to updated scores, while KISSAT adopts the propagating definition, calling it “glue” in code.

■ **Table 6** Number of solved instances for different configurations w/ and w/o the *used* flag.

configuration	+ used	configuration	+ used	configuration	+ used
critical@size≤10	322 328 (+6)	critical@size≤6	319 322 (+3)	critical@lbd≤3	315 324 (+9)
rank@size=50%	310 322 (+12)	rank@lbd=50%	312 325 (+13)	activity	290 294 (+4)

We evaluate LBD and critical clause ideas independently (Fig. 4). One set of experiments compares different *critical* thresholds (Alg. 1, line 7) across target fractions using *rank* (line 10); another uses clause size instead. Although the average size/LBD ranged from 2.05 to 2.15, we expected *glue*≤5 to behave like *size*≤10—but it did not.

Still, surprisingly, LBD and size are nearly interchangeable. Retaining clauses up to size 10 yields the best results, while *rank* configurations trail the best *critical* ones. This suggests that setting an appropriate critical threshold suffices to identify valuable clauses, whereas *rank* tends to retain more unhelpful ones, hurting performance.

We also evaluated combinations of *critical* and *rank* (Fig. 5) by enabling both lines 7 and line 10 in Alg. 1. Since critical clauses are always retained, we reverted to the default unlearn target of $f = 75\%$. This significantly improved performance for lower thresholds, such as *lbd*≤2, *lbd*≤3 or *size*≤6, with smaller gains for the higher *size*≤10 threshold.

6 Used Clauses are Useful

In 2019, Soos et al. [27] and, more recently, Yang et al. [29] applied machine learning techniques to identify effective criteria for clause usefulness. Their approach involved running the solver without performing any clause unlearning while logging detailed statistical data for each learned clause. Subsequently, a proof checker trimmed the proof and extracted resolution chains using a heuristic that favored reusing existing clauses over introducing new ones during derivation. Their analysis confirms that the LBD is an important predictor of a clause’s usefulness—with LBD slightly outperforming clause size—and demonstrated that a clause’s recent use in deriving a conflict is a strong estimator of its future utility.

Various solvers, including CADI₁CAL, implement a *used* flag for clauses, set during conflict analysis and reset during unlearning (*cf.* Alg. 1). In essence, the *used* flag reduces clause activity to a single bit recording whether the clause was used since the last unlearning. It is very similar to the *second chance algorithm* or *clock algorithm* [26] used in page replacement policies, as it gives the clause a second chance to be useful before unlearning.

We combine this *used* flag with each of the configurations from the previous sections and compare the difference in the solved instances when retaining *all* used clauses (+ *used*) versus the configuration without *used* flag (Tab. 6). Subsequently, we refer to these configurations as “<configuration>+used”. Keeping only used clauses performs slightly better than activity-based unlearning, showing that it is indeed a useful abstraction (Fig. 9). However, it still performs worse than the baseline. The full potential of the *used* flag is only realized in conjunction with other unlearning methods: it *consistently* improves each configuration’s performance. Hence, used clauses are important, but less so than critical clauses.

The *used* flag inspired us to give clauses one more chance, namely a *third* chance. When clauses are used, we set *used* to 2 and decrement it during unlearning. We consider the clause for deletion only when we reach zero. This in essence simulates a three-tier system [23], but in our experiments does not yield any improvements (see Fig. 9 in the appendix).

Another interesting question is how to update the *used* flag during inprocessing. We recommend *not* updating the *used* flag during forward subsumption, as doing so resulted in significant performance degradation in CADI₁CAL. Specifically, in version 1.9.4 we altered

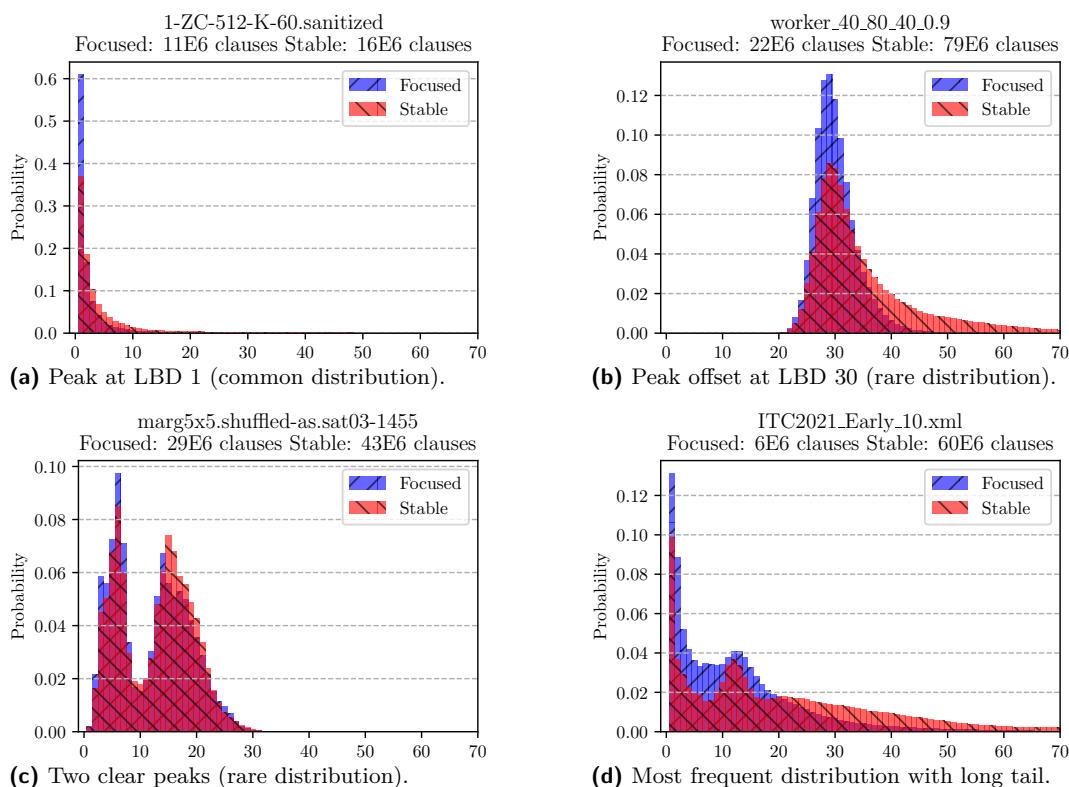


Figure 7 LBD distribution for selected benchmarks in *no-reduce* (keep all clauses). Benchmark name is shown above each graph together with how many clauses were learned during the run.

the policy for updating the *used* flag on successfully strengthened learned clauses—changing it from granting a second chance to providing a third chance. This modification led to a severe performance regression in incremental solving, and was reverted in CADICAL 1.9.5.

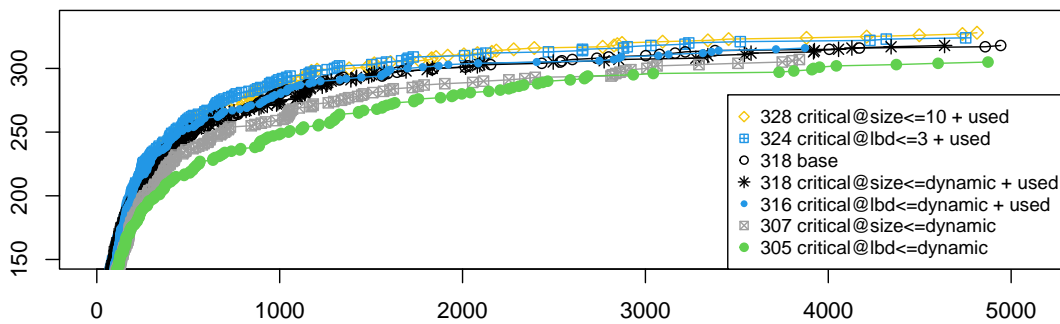
7 Odd LBD Distributions and Dynamic Tier Calculations

In order to link LBD to usefulness, we further recorded the LBD of all clauses used during conflict analysis on all benchmarks from the SAT Competition 2024. Note that KISSAT runs a portfolio of two modes; *focused* mode (many restarts, shallow exploration of the search space), and *stable* mode (few restarts, and deep exploration of the search space with long assignments). Chanseok Oh [23] calls these modes “UNSAT” and “SAT” modes. Due to these different search characteristics, we recorded LBD statistics for both modes separately.

We show 4 representative instances from the SAT Competition 2024 in Fig. 7. The most common LBD distribution shows a sharp peak at very low LBD values, followed by a steep decline. However, some problem families exhibit a broader distribution, or the peak is shifted to much higher LBD values. In certain cases, there are *no* used clauses with low LBDs at all. As a result, *no* critical clauses would be retained unconditionally. In such situations, unconditionally keeping *used* clauses (and ranking) helps to compensate for this issue.

The skewed LBD distributions motivated us to replace the static tier limit for identifying critical clauses with a dynamic approach based on either LBD or size. Specifically, we set the threshold so that 50% of all clauses used since the beginning fall below this limit.

We also observed some instances where most learned clauses have an LBD of 1. In such



■ **Figure 8** Runs with dynamically calculated tier 1 limit (50% clauses below the limit were used).

cases, using a tier threshold of LBD 2—typically seen as very good—would actually be problematic, as it would prevent those clauses from ever being unlearned.

The dynamic unlearning scheme (Fig. 8) does not achieve the same performance as the static schemes, suggesting that the usefulness of critical clauses comes from the absolute size or LBD, and not the comparatively low size or LBD. The performance of dynamic configurations is similar to the non-dynamic configurations on satisfiable instances but notably worse on unsatisfiable instances. This, surprisingly, indicates that in the dynamic configurations too few useful clauses are kept.

The *used* flag of the previous experiment also improves dynamic configurations. Further, size and LBD in combination with dynamic tier limit computation perform similarly. Nonetheless, the dynamic configurations together with the *used* flag still perform worse. While we see as usual that the *used* flag improves the performance and that size is similar to LBD, the resulting configuration is still worse than the version where we keep all clauses of size 10. We have observed during earlier runs on SAT Competition 2023 benchmarks that there were more outliers for the glue distributions compared to SAT Competition 2024 benchmarks. We leave the investigation of this observation as future work.

8 Conclusion and Future Work

We investigated which unlearning schemes are useful for the state-of-the-art SAT solver KISSAT on SAT Competition 2024 instances. Our results show that, overall, activity performs the worst. Furthermore, while the difference between LBD and size based ranking is limited, it is important to keep either low LBD or low size clauses unconditionally, as well as *used* clauses, which consistently gives a boost for all our considered unlearning strategies. A valid configuration which could replace the current strategy in KISSAT could therefore be `critical@size<=6 + rank@size=75% + used`. Probably the most striking result is that frequently unlearning all clauses completely is in essence not harmful for solving satisfiable instances.

As future work it would be interesting to understand whether these findings can be extended to other SAT solvers and persist on other benchmark sets, as the diversity of solved instances for the strategies is perhaps limited by the benchmark set (the virtual best solves 351 instances, only 22 more than the single best strategy).

References

- 1 Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *IJCAI*, pages 399–404, 2009.
- 2 Gilles Audemard and Laurent Simon. Refining restarts strategies for SAT and UNSAT. In *CP*, volume 7514 of *Lecture Notes in Computer Science*, pages 118–126. Springer, 2012.
- 3 Gilles Audemard and Laurent Simon. On the Glucose SAT solver. *Int. J. Artif. Intell. Tools*, 27(1):1840001:1–1840001:25, 2018. doi:10.1142/S0218213018400018.
- 4 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL, Gimsatul, IsaSAT and Kissat entering the SAT Competition 2024. In Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2024 – Solver, Benchmark and Proof Checker Descriptions*, volume B-2024-1 of *Department of Computer Science Report Series B*, pages 8–10. University of Helsinki, 2024.
- 5 Armin Biere, Matti Järvisalo, Daniel Le Berre, Kuldeep S. Meel, and Stefan Mengel. The SAT practitioner’s manifesto, September 2020. doi:10.5281/zenodo.4500928.
- 6 Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962. doi:10.1145/368273.368557.
- 7 Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960. doi:10.1145/321033.321034.
- 8 Rina Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, 1990.
- 9 Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003. doi:10.1007/978-3-540-24605-3_37.
- 10 Jan Elffers, Jesús Giráldez-Cru, Stephan Gocht, Jakob Nordström, and Laurent Simon. Seeking practical CDCL insights from theoretical SAT benchmarks. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 1300–1308. ijcai.org, 2018. doi:10.24963/IJCAI.2018/181.
- 11 Roman Gershman and Ofer Strichman. Haifasat: a SAT solver based on an abstraction/refinement model. *J. Satisf. Boolean Model. Comput.*, 6(1-3):33–51, 2009. doi:10.3233/SAT190061.
- 12 Matthew L Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.
- 13 Matthew L Ginsberg. Satisfiability and systematicity. *Journal of Artificial Intelligence Research*, 53:497–540, 2015.
- 14 Eugene Goldberg and Yakov Novikov. BerkMin: A fast and robust Sat-solver. *Discret. Appl. Math.*, 155(12):1549–1561, 2007. doi:10.1016/J.DAM.2006.10.007.
- 15 Evgenii I. Goldberg and Yakov Novikov. Berkmin: A fast and robust Sat-solver. In *2002 Design, Automation and Test in Europe Conference and Exposition (DATE 2002), 4-8 March 2002, Paris, France*, pages 142–149. IEEE Computer Society, 2002. doi:10.1109/DATE.2002.998262.
- 16 Bernhard Gstrein, Florian Pollitt, André Schidler, Mathias Fleury, and Armin Biere. Source code kissat-cr. Each configuration in the paper is one branch. URL: <https://github.com/texmex76/kissat-cr>.
- 17 Bernhard Gstrein, Florian Pollitt, André Schidler, Mathias Fleury, and Armin Biere. Learn to unlearn, 04 2025. doi:10.5281/zenodo.15146408.
- 18 Said Jabbour, Jerry Lonlac, Lakhdar Sais, and Yakoub Salhi. Revisiting the learned clauses database reduction strategies. *International Journal on Artificial Intelligence Tools*, 27, 02 2014. doi:10.1142/S0218213018500331.

- 298 **19** Matti Järvisalo, Marijn Heule, and Armin Biere. Inprocessing rules. In Bernhard Gramlich,
299 Dale Miller, and Uli Sattler, editors, *Automated Reasoning - 6th International Joint Conference,*
300 *IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes*
301 *in Computer Science*, pages 355–370. Springer, 2012. doi:10.1007/978-3-642-31365-3_28.
- 302 **20** Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-
303 world SAT instances. In Benjamin Kuipers and Bonnie L. Webber, editors, *Proceedings of the*
304 *Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications*
305 *of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode*
306 *Island, USA*, pages 203–208. AAAI Press / The MIT Press, 1997. URL: [http://www.aaai.](http://www.aaai.org/Library/AAAI/1997/aaai97-032.php)
307 [org/Library/AAAI/1997/aaai97-032.php](http://www.aaai.org/Library/AAAI/1997/aaai97-032.php).
- 308 **21** João Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers.
309 In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of*
310 *Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 133–182.
311 IOS Press, 2nd edition, 2021. doi:10.3233/FAIA200987.
- 312 **22** Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik.
313 Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation*
314 *Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535. ACM, 2001.
315 doi:10.1145/378239.379017.
- 316 **23** Chanseok Oh. Between SAT and UNSAT: the fundamental difference in CDCL SAT. In Marijn
317 Heule and Sean A. Weaver, editors, *Theory and Applications of Satisfiability Testing - SAT*
318 *2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*,
319 volume 9340 of *LNCS*, pages 307–323. Springer, 2015. doi:10.1007/978-3-319-24318-4_23.
- 320 **24** João P. Marques Silva and Karem A. Sakallah. GRASP - a new search algorithm for
321 satisfiability. In Rob A. Rutenbar and Ralph H. J. M. Otten, editors, *Proceedings of the*
322 *1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, San*
323 *Jose, CA, USA, November 10-14, 1996*, pages 220–227. IEEE Computer Society / ACM, 1996.
324 doi:10.1109/ICCAD.1996.569607.
- 325 **25** Laurent Simon. Post mortem analysis of SAT solver proofs. In Daniel Le Berre, editor, *POS-14.*
326 *Fifth Pragmatics of SAT workshop*, volume 27 of *EPiC Series in Computing*, pages 26–40.
327 EasyChair, 2014. doi:10.29007/gpp8.
- 328 **26** Alan Jay Smith. Sequentiality and prefetching in database systems. *ACM Trans. Database*
329 *Syst.*, 3(3):223–247, 1978. doi:10.1145/320263.320276.
- 330 **27** Mate Soos, Raghav Kulkarni, and Kuldeep S. Meel. CrystalBall: Gazing in the black box
331 of SAT solving. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of*
332 *Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal,*
333 *July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages
334 371–387. Springer, 2019. doi:10.1007/978-3-030-24258-9_26.
- 335 **28** Richard M Stallman and Gerald J Sussman. Forward reasoning and dependency-directed
336 backtracking in a system for computer-aided circuit analysis. *Artificial intelligence*, 9(2):135–
337 196, 1977.
- 338 **29** Jiong Yang, Arijit Shaw, Teodora Baluta, Mate Soos, and Kuldeep S. Meel. Explaining SAT
339 solving using causal reasoning. In *SAT*, volume 271 of *LIPICs*, pages 28:1–28:19. Schloss
340 Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

A Appendix: More Details on Unlearning in Kissat

A simplified version of the unlearning algorithm implemented in KISSAT is shown in Fig. 1 in Sect. 2. Below, we elaborate on aspects omitted from the simplified exposition. In our experiments, these aspects were either disabled in the evaluated **base** version or remained unchanged throughout the study.

A key aspect not yet discussed is the scheduling of the *unlearn* function—the limit on conflicts and learned clauses required before triggering the next *unlearn* call. In MINISAT, this interval increases *geometrically* (e.g., starting at 10k conflicts and growing by 50% each time). The GLUCOSE paper [1] suggests adopting an *arithmetic* schedule by retaining critical clauses. For example, one may start unlearning after 1000 conflicts and increment the interval by a fixed amount (e.g., 300 conflicts per round), lowering the frequency of unlearning.

Under such an arithmetic scheme, the conflict interval between the n -th and next *unlearn* is $300 \cdot n + 1000$, or more generally $period \cdot n + initial$. Yet, even assuming a constant target fraction ($f = 50\%$ in GLUCOSE) of removed clauses and no tier system, analyzing its effect on the expected number of retained clauses after n rounds is non-trivial. The **competition** version instead schedules unlearning after $period \cdot \sqrt{n} + initial$ conflicts, with both parameters set to 1000—diverging from GLUCOSE’s arithmetic scheme. As scheduling adds another investigative dimension, we leave it unchanged and defer further analysis to future work.

In the **competition** configuration version of KISSAT, the target fraction f increases dynamically from 50% to 90%, scaling logarithmically with the number of conflicts. The **base** version uses a constant $f = 75\%$ and lacks this dynamic update.

Additionally, the **competition** version adopted a three-tier system, where clauses in tier 3 (e.g., $LBD > 6$) get only a second chance to survive unlearning, while medium-critical clauses (e.g., $LBD > 2$) receive a third chance, as in Sect. 6. The **base** version omits the extended *used* range for critical clauses in tier 1. This encoding allows even critical clauses ($LBD \leq 2$) to be eventually unlearned if unused across many rounds. In **competition**, 5 bits encode *used*, so such clauses are removed after 31 unlearning rounds without use. In contrast, **base** sets *used* to 2, limiting the range to three values.

Moreover, like many other solvers, the **competition** configuration does *not* restart (i.e., backtrack to level zero) during unlearning (see also Sect. 3). Thus, clauses used as reasons on the trail must be retained. Such clauses, along with subsumed and root-level satisfied ones, are skipped in the loop in Fig. 1. Since restarts are more frequent in focused mode than stable mode, average conflict levels are lower in focused mode. Not restarting during unlearning thus increases the chance of retaining reason clauses in stable mode. To avoid this bias, the **base** version performs a restart before unlearning.

Finally, as discussed in Sect. 7—the study’s starting point—the default KISSAT version *dynamically* computes LBD limits for tiers. However, despite diverse observed distributions, we found no measurable benefit from this technique.

B Appendix: Additional Experiments

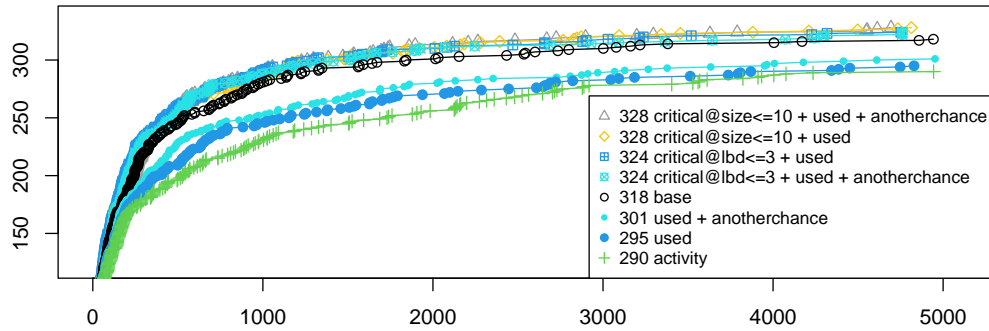


Figure 9 Comparing the effect of used, activity and the effect of giving clauses another chance.

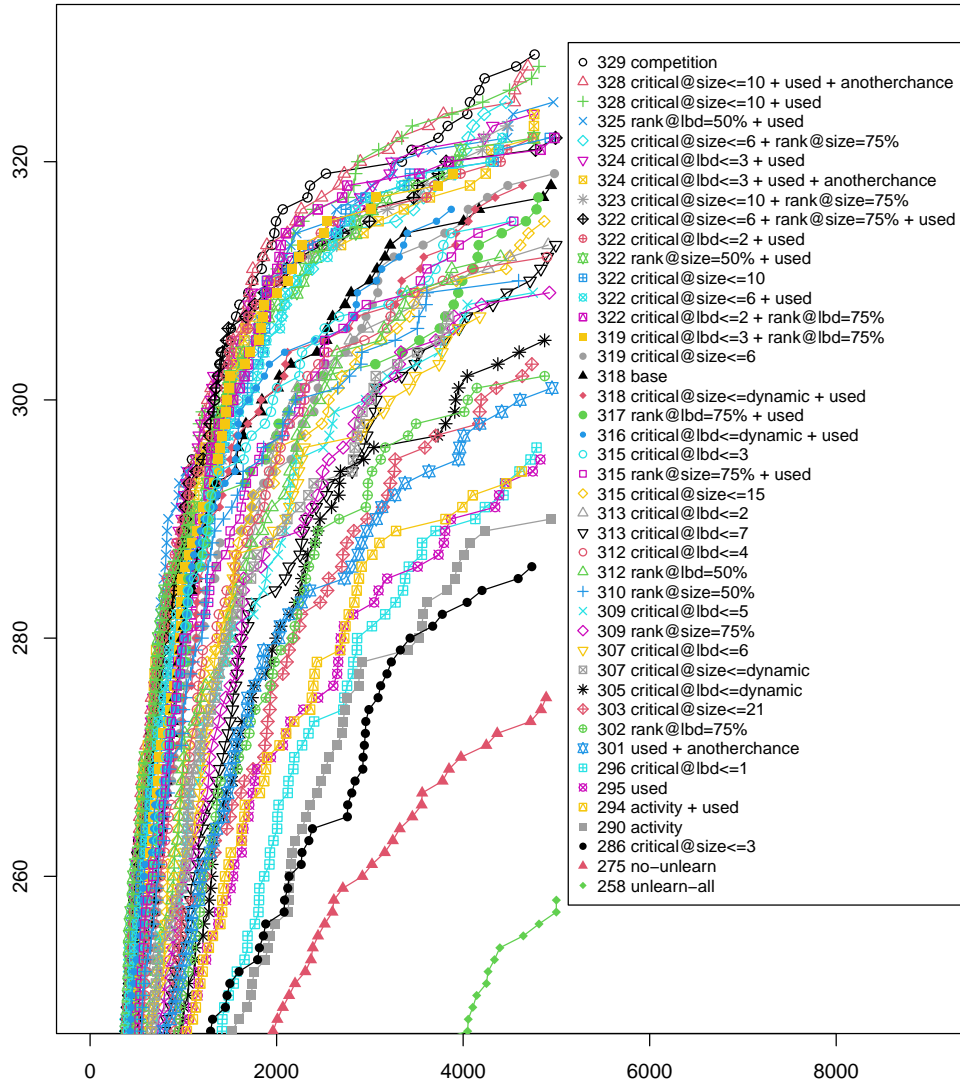


Figure 10 Cumulative distribution function (CDF, 100% = 400 instances) of all interesting runs.