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 an essential part of the CDCL algorithm and a significant milestone in the development of SAT solving. However, keeping all learned clauses without discrimination turns CDCL into an exponential space algorithm, gradually slowing 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 SAT solver Kissat [3]. To decrease potential influences on the unlearning strategy, we develop a simplified base version of Kissat, which acts as the starting point for our experiments. Our goal is to explore simple ideas first and gradually increase complexity by combining orthogonal approaches.

Our first experiment, perhaps unsurprisingly, shows that retaining all clauses impacts performance negatively across all instances. However, for satisfiable instances, periodically removing all learned clauses yields near state-of-the-art performance. For unsatisfiable instances, it is vital to always keep some learned clauses.

After having explored these all-or-nothing strategies, we consider keeping a certain percentage of learned clauses. We test different Ranking functions: clause size, Literal Block Distance (LBD) and clause activity. Clause activity is a combination of how often and how recently clauses are used during conflict analysis, introduced by Minisat [4], while LBD refers to the number of levels the learned clause spans at the time of learning, an essential idea of the Glucose Solver [2]. Our experiments show that both size- and LBD-based rankings outperform activity by a big margin, but are comparable to each other.

We continue with the second idea of the influential Glucose paper [1] concerning unlearning, i.e., using a threshold instead of a ranking function, meaning that all clauses below the threshold are kept instead of a certain percentage. Replacing our size and LBD rankings yields improvements for certain thresholds, for both size and LBD.

To turn clause activity into a threshold, we look at the "used" abstraction which only requires one bit, namely weather the clause has been used since the last unlearning phase. We get comparable performance for activity and used in our experiments. Extending the abstraction to more bits does not improve performance significantly.

Moving to combined strategies, we focus on the more successful standalone configurations and orthogonal ideas. Combining ranking and threshold for size and LBD gives another performance increase, while used consistently improves all tested combinations. Most successful are combinations with a size or LBD threshold, together with either a glue threshold or a size or LBD ranking.

Last but not least we explore the idea of dynamically-computed instead of fixed LBD thresholds. We follow the intuition that LBD values of recently used clauses are relevant and compute a dynamic threshold based on these clauses. Unfortunately, the resulting clause unlearning strategy performs worse than the simpler strategies above, even though LBD distribution charts suggest that at least for some benchmarks optimal thresholds should differ from the default.

In conclusion, we can say that revisiting clause unlearning gave some interesting insights, identifying potentially good and simple strategies with similar performance to the original solver.

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` [5]
Dataset (Log Files): `https://doi.org/10.5281/zenodo.15146408` [6]

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.

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. On the Glucose SAT solver. *Int. J. Artif. Intell. Tools*, 27(1):1840001:1–1840001:25, 2018. `doi:10.1142/S0218213018400018`.

3 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.

4 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`.

5 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`.

6 Bernhard Gstrein, Florian Pollitt, André Schidler, Mathias Fleury, and Armin Biere. Learn to unlearn, 04 2025. `doi:10.5281/zenodo.15146408`.