MallobSat naps in the SAT Competition 2025

Dominik Schreiber Niccolò Rigi-Luperti
Institute of Information Security and Dependability
Karlsruhe Institute of Technology
Karlsruhe, Germany
{dominik.schreiber , niccolo.rigi-luperti}@kit.edu

Armin Biere
Chair of Computer Architecture
University of Freiburg
Freiburg i.Br., Germany
biere@cs.uni-freiburg.de

***Abstract*—We present the 2025 version of the parallel and distributed SAT solver MallobSat to participate in this year's parallel track of the SAT Competition.**

***Index Terms*—SAT solving, HPC, preprocessing**

I. Introduction

Since SAT Competition 2025 does not feature a massively parallel / distributed / cloud track, the distributed solving features of MallobSat [7] remain un(con)tested this time – hence one might say that our system takes a nap this year. Still, we prepared a lightweight submission of single-process MallobSat to the parallel track. We use the same general setup and process layout as MallobSat in last year's parallel track [4] and integrated the 2024 solver backend of Kissat [2].

II. Naps

We introduce a new solving approach we call "naps" ("*not a portfolio solver!*"). The central features of this setup are:

- We run four YalSAT [1] local search solvers and $p-4 = 60$ *uniform* Kissat solver threads. Across all threads, all pre– and inprocessing features are switched off, as well as the solver-specific ("native") cycled diversification of earlier years. The only kind of diversification we employ for the Kissat threads is very light: setting different random seeds and sparsely setting some random variable phases. This setup clearly departs from a classical "solver portfolio" and, in our view, should rather be considered a uniform parallel CDCL procedure with clause sharing acting as search space pruning. The few additional local search threads serve as independent "needle-in-the-haystack" searchers for a satisfying assignment.
- We employ a singular a-priori sequential preprocessor (not unlike the BVA performed in PL-PRS-BVA-KISSAT [3]), which uses Kissat's full preprocessing arsenal, as well as some preprocessing in Lingeling (Cardinality Constraint reasoning and Gaussian elimination). This allows to at times immediately recognize (un)satisfiability or, otherwise, simplify the formula. While this preprocessing is running, we already run a solving task at all (other) threads on the original formula. When preprocessing is done, we evict this solving task and replace it with a solving task that operates on the preprocessed formula instead. Any model reported by the latter task is transformed back into a model for the original formula.
- We replace MallobSat's strategy of incrementing incoming clauses' LBD values [7] with the simpler strategy of resetting all LBD values of clauses to their maximum once they are produced. This functionally disables all roles of LBD values in our clause sharing – in recent experiments we did not observe any loss in performance this way, while it simplifies the sharing logic.

We describe and evaluate (a slightly earlier version of) this setup in depth in our SAT'25 publication [6]. Note that Kissat is the only CDCL implementation we employ – we only use Lingeling's preprocessing and local search (YalSAT). As a side effect, this reduces the active code, simplifying maintenance and streamlining future research (also and especially in the distributed setting). We do not submit a version of our real-time proof-checking setup [4, 5] since the preprocessing does not yet support proof production. As a next step, we intend to replace our current preprocessing with proof-producing preprocessing and integrate it in our trusted solving pipeline.

Acknowledgment

The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer SuperMUC-NG at Leibniz Supercomputing Centre (www.lrz.de). Research reported in this work was supported by an Amazon Research Award (Fall 2023).

References

[1] Armin Biere. "Yet another local search solver and Lingeling and friends entering the SAT Competition 2014". In: *SAT Competition*. 2014, p. 65.

[2] Armin Biere et al. "CaDiCaL, Gimsatul, IsaSAT and Kissat entering the SAT Competition 2024". In: *SAT Competition 2024: Solver, Benchmark and Proof Checker Descriptions* (2024), pp. 8–10.

[3] Mazigh Saoudi et al. "PL-PRS-BVA-KISSAT in SAT Competition 2024". In: *Pragmatics of SAT*. 2024. URL: https://dl.lre.epita.fr/papers/saoudi.24.pos.pdf.

[4] Dominik Schreiber. "MallobSat and MallobSat-ImpCheck in the SAT Competition 2024". In: *SAT Competition 2024: Solver, Benchmark and Proof Checker Descriptions* (2024), pp. 22–23.

[5] Dominik Schreiber. "Trusted Scalable SAT Solving with on-the-fly LRAT Checking". In: *Theory and Applications of Satisfiability Testing (SAT)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 2024, 25:1–25:19. DOI: 10.4230/LIPIcs.SAT.2024.25.

[6] Dominik Schreiber, Niccolò Rigi-Luperti, and Armin Biere. "Streamlining Distributed SAT Solver Design". In: *Theory and Applications of Satisfiability Testing (SAT)*. 2025, 23:1–23:23. DOI: 10.4230/LIPIcs.SAT.2025.23.

[7] Dominik Schreiber and Peter Sanders. "MallobSat: Scalable SAT Solving by Clause Sharing". In: *Journal of Artificial Intelligence Research* 80 (2024), pp. 1437–1495. DOI: 10.1613/jair.1.15827.