Practical SAT Solving (ST 2025) | Assignment 4

Markus Iser, Dominik Schreiber, Niccolò Rigi-Luperti
Algorithm Engineering & Scalable Automated Reasoning (KIT) 2024-06-17 – 2024-07-01

1 Competition: Clause Sharing Selection Heuristic (3(+7) Points)

Manipulate Mallob's clause sharing by modifying the class `ClausePrefilter` (see below), whose method `prefilterClause` is called whenever a solver thread produces a clause that passes a number of basic checks. The method can return `false` to completely drop a clause; if the method returns `true`, the calling solver thread will subsequently attempt to write the clause to an export buffer. You can change a (non unit) clause's priority by manipulating its LBD value to make it more or less likely that a clause "survives" subsequent selection stages and is shared successfully. For a clause of length $\ell \geq 2$, LBD values of 2 ("best") through ℓ ("worst") are valid. If needed, you may perform precomputations and access the formula in the body of `notifyFormula(·)`.

You get 3 points for submitting a functional version of `clause_prefilter.hpp` with some non-trivial change. The submissions that result in the best performance with a Kissat portfolio at 64 cores on diverse benchmarks get up to 7/5/3 additional points. We will, in particular, set the non-default parameters `-lbdpi=1 -lbdpo=1 -pbbm=1` to consistently prioritize clauses by "LBD" first and by clause length second (i.e., as a tiebreaker only) and `-rlbd=1 -ilbd=0` to reset each incoming clause's LBD value to ℓ before a solver thread imports it.

Mallob version to use: `https://github.com/domschrei/mallob/tree/experimental`
— `ClausePrefilter`: see `src/app/sat/sharing/filter/clause_prefilter.hpp`

Contact Dominik ⟨dominik.schreiber@kit.edu⟩ in case of technical questions or issues.

2 Parallel Resolution (3 + 3 Points)

Consider a parallel setup with p resolution procedures in a "bulk synchronous parallel" (BSP) model: Each procedure performs a single resolution step; then, all resolvents are shared across all procedures. This is repeated until the empty clause is found.

(a) Provide an unsatisfiable CNF formula where the above setup *can* speed up the derivation of the empty clause by a factor of $p/\log p$ (or better) compared to the shortest sequential resolution order.

(b) Provide an unsatisfiable CNF formula where the derivation of the empty clause can only ever be sped up by a factor of $1 + \varepsilon$ (where $0 \leq \varepsilon \ll 1$) compared to the shortest sequential resolution order.

3 VBS and Speedups (4 + 6 Points)

Analyze the results of the 2024 SAT Competition.

(a) Compute a VBS of all sequential main track solvers. Compare its performance to the best cloud track solver in the competition. Interpret the results, also considering any peculiar instance families.

(b) Compute the geometric mean and median speedup of each parallel track solver relative to the best performing sequential solver `kissat-sc2024`, only considering instances which both the sequential solver *and* the parallel solver were able to solve. Then do the same while considering *all* instances solved by the parallel solver – "generously" attributing a running time of 5 000 s to each such instance that the sequential solver did not solve. Interpret and compare the results.

Competition results: `https://satcompetition.github.io/2024/downloads/detailed_results.zip`
Benchmark meta data: `https://satcompetition.github.io/2024/downloads/meta.csv`

4 Clause Filtering in Distributed SAT (5 Points)

Distributed clause-sharing solvers like MallobSat filter a repeated clause based on *exact syntactical equivalence* (i.e., if a clause c is shared successfully, then clause c will be blocked for some period). Find a way to generalize this approach to *subsumed* clauses, i.e., if clause c is shared, then all clauses $c' \supseteq c$ are blocked for some period. Provide a rough analysis of the running time complexity and the memory footprint of your approach.