Implementation

My initial solver implemented unit propagation with chronological backtracking. For decisions, it split on literals in the order they appeared in the input. As expected, this was correct on the toy instances but performed poorly on the benchmark suite (commit `fd4a11c`; `logs/log-fd4a11c`).

Instance	Time
U50_1065_045.cnf	250.31

Next, I replaced the standard library `HashSet` with `indexical`, which represents sets as bitsets. On `U50_1065_045.cnf` this reduced runtime by roughly 75% (commit `d5f6f8c`; `logs/log-d5f6f8c`).

Instance	Time
C1065_064.cnf	62.50
U50_1065_045.cnf	59.19

I then implemented two-watched literals for propagation. This substantially reduced propagation overhead and allowed the solver to complete roughly six additional benchmarks (commit `530d90b`; `logs/log-530d90b`).

After that, I improved the decision heuristic by ordering literals by their frequency in the input clauses. This adds a one-time preprocessing pass to count occurrences, but it makes decisions more informed than a pure "first-seen" rule. This change enabled a few more benchmarks to finish within the time limit (commit `712453`; `logs/log-7124535` and `logs/log-7124535-2`).

At that point, I noticed timeouts on instances that other solvers handled quickly (often under 1 second). The missing piece was preprocessing with unit propagation and pure-literal elimination at the start. Adding these steps dramatically improved performance and brought several previously timing-out instances into the seconds range, including `C140.cnf`, `C208_3254.cnf`, `C210_30.cnf`, and `C210_55.cnf` (commit `c8d952a`; `logs/log-c8d952a`).

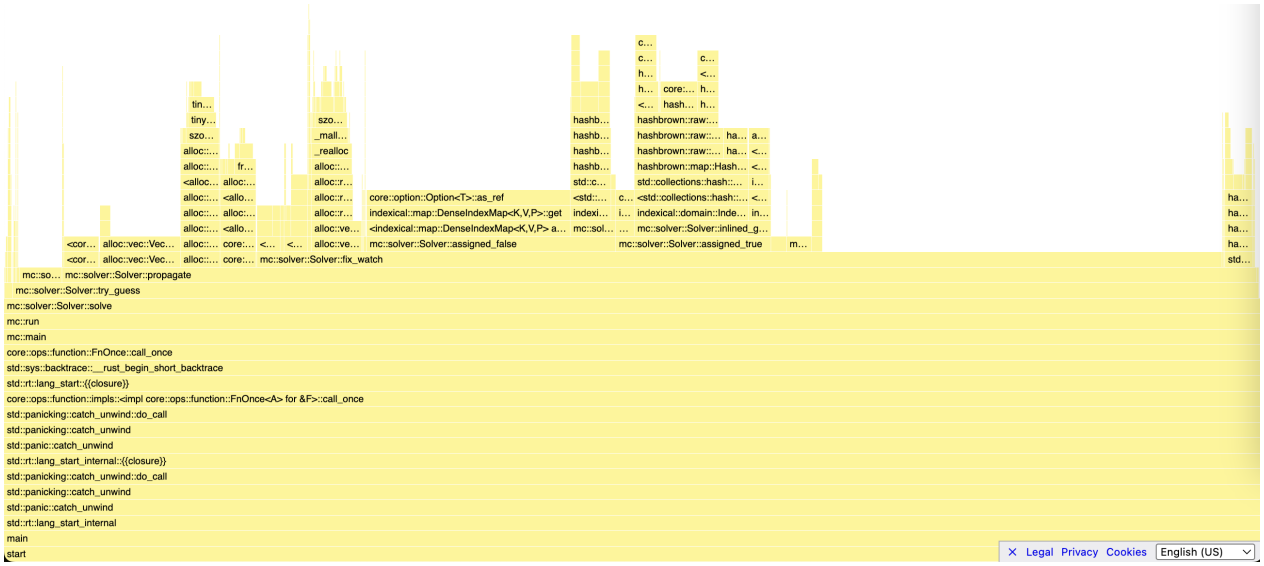
Instance	Time
C1065_064.cnf	1.23
C1065_082.cnf	1.01
C1597_024.cnf	7.84
C1597_060.cnf	10.00
C1597_081.cnf	1.24
C168_128.cnf	0.03
C181_3151.cnf	0.02
C208_120.cnf	0.17
C459_4675.cnf	0.00
U50_1065_038.cnf	0.75
U50_1065_045.cnf	1.19
U75_1597_024.cnf	4.41

I also experimented with a few micro-optimizations:

1. Circular search for the next watched position (commit `6524b9b`)
2. Switching from `HashMap` to `IntMap` (commit `a7b2944`)
3. Adding inline hints on hot functions (commit `a7b2944`)

These changes did not produce a measurable runtime improvement.

Finally, I attempted to implement CDCL with non-chronological backjumping. This version was significantly slower and also failed several instances that the DPLL-style solver could solve. Profiling on `U50_4450_035.cnf` showed that `get_vidx` was on the hot path, suggesting the overhead of my current representation (in particular, frequent bitset/index conversions) dominated runtime.



I also implemented VSIDS on top of CDCL, but this further slowed the solver. My working hypothesis is that the per-conflict score updates (and heap maintenance) outweighed any benefit from better branching decisions in my current implementation.

Reflection

I think the data structure representation is much more important than I realized. I used `indexical`, which allows you define a finite domain of objects and assign a numeric index to each object. I chose this library because literals may be non-consecutive (e.g. Even if there are 50 literals, it is not necessary 0 to 49 being used.) However, I think the underlying bitset implementation is not very efficient, and it may be better to just preallocate an array and use that instead. This also could have made CDCL, VSIDS, and restarts more viable.

I also think there are a lot of better splitting heuristics that I could have implemented.

In total I spent 40.5 hours on the project.