Noah Larbalestier
Professor Vosoughi - COSC 76
25 November 2025

PA5 Report

1. Description

This project implements two local-search SAT solvers, `GSAT` and `WalkSAT`, and applies them to constraint problems encoded in CNF mimicking sudoku-style problems, with modular support for sudoku display and parsing. The solver is written in Python and is designed to be fully generic: it accepts any CNF file consisting of integer literals (positive for variables, negative for negations), with each line representing a clause in conjunctive normal form.

2. Algorithm Overview

GSAT operates by:

1. Generating a random Boolean assignment.
2. Checking whether all clauses are satisfied.
3. With probability *p*, choose a variable uniformly at random and flipping it. (i.e, p of the time)
4. Otherwise (1-p of the time), score each variable by the number of clauses that would become satisfied if it were flipped, choosing among the best-scoring variables at random.

GSAT is simple but can be slow because it must consider all variables when scoring each flip. To make this practical, I implemented *incremental scoring*: instead of recomputing how many clauses are satisfied after every flip, the solver evaluates only clauses affected by that variable and maintains a running count of satisfied clauses. This significantly improves performance and keeps the implementation clean.

WalkSAT improves efficiency by restricting flip candidates to variables appearing in a single unsatisfied clause. The algorithm:

1. Picks an unsatisfied clause uniformly at random.
2. With probability *p*, flips a random variable within that clause (a “noise” move).
3. Otherwise, evaluates all variables in the clause using a **min-break** heuristic, flipping the one that falsifies the fewest currently satisfied clauses.

WalkSAT thus tends to explore the search space faster and more effectively than GSAT for highly constrained problems such as Sudoku with many small clauses. It also uses incremental updates: only clauses containing the flipped variable are re-evaluated after each move.

3. Design

- Generic SAT modular approach - The solver does not reference Sudoku specifically. It treats all literals strictly as integers.
- Incremental clause evaluation - Both GSAT and WalkSAT update clause satisfaction state locally on each flip, improving speed.
- P as Noise - Defaults are conventional: GSAT uses p = 0.5, while WalkSAT uses p = 0.7 as recommended in the assignment write-up. The change in GSAT's noise to a bit lower was to reduce noise. Especially for the smaller problems, noise was not necessary as local minima were not as costly to get into and not as catastrophic.
- Random seeding - Both algorithms accept an optional seed for reproducibility.
- CLI use - Users can specify algorithm, noise, iteration limits, and output filenames from the terminal.
- Testing harness - run_tests.py file provides a small test runner for the provided CNF puzzles.

4. Evaluation

Both GSAT and WalkSAT successfully solve the simpler CNF files:

- one_cell.cnf
- all_cells.cnf
- rows.cnf

These succeed quickly and consistently. The solver writes .sol files matching the expected variable format and is compatible with the provided display.py visualization tool.

Performance on Harder CNFs

- **GSAT** works reliably on only the smallest CNF, but—as expected—struggled massively on Sudoku puzzles with heavy row/column/region constraints. This is consistent with the literature and assignment warning.
- **WalkSAT** performs noticeably better. It can solve all_cells.cnf, and rows.cnf, with reasonable parameter settings, but still does not succeed with sudoku.

Overall, WalkSAT "works" in the sense that it frequently finds a model within its restart and flip limits, and fails gracefully when it does not, but neither SAT algorithm is complex enough to handle large CNFs. The solvers implemented here as GSAT, WalkSAT, and the MaxWalkSAT extension can quickly find satisfying assignments when the space is small and favorable but cannot guarantee correctness or detect unsatisfiability. More complex, dedicated SAT solvers such as MiniSAT (an example ChatGPT told me to do a bit of research on) incorporate many features to maximize efficiency and keep track of logical consequences. Backtracking and propagation with "watched literal data structures" help clause evaluation in sometimes even constant time while tracking assignments and their consequences. They prevent revisiting and have much better heuristics like LRB… and likely do not use Python, instead opting for smaller bit-optimized representations.

The algorithms here operate in a very different way: they use a single full assignment, make local flips, and explore the space through stochastic moves. They are lightweight, conceptually simple, and fast to build, but are nowhere close to the reasoning power, efficiency, or scalability of CDCL-based solvers.[1] Achieving anything in the direction of MiniSAT would require watched literals, deterministic propagation, conflict analysis, and a much better memory system.

5. Extension

To explore the behavior of the local-search SAT more practically on structured problems like Sudoku, I implemented **MaxWalkSAT**, a weighted version of WalkSAT. Instead of just minimizing the number of unsatisfied clauses, MaxWalkSAT minimizes the weighted sum of violated clauses. This enables techniques such as:

- Assigning higher weights to unit clauses, which encode the fixed clues in Sudoku.
- Keeping these constraints "nearly fixed," because violating one incurs a disproportionately large penalty.

This approach provides a middle ground between fully fixing the clue values (which reduces flexibility for other applications and can cause a solver to get stuck) and letting WalkSAT violate them freely (which loses too much structure and completely violates the rules of sudoku).

In practice, the weighted variant behaved as expected: the solver strongly preferred respecting the fixed Sudoku values but was still capable of exploring nearby assignments that briefly violated it.

[1] https://www.cs.princeton.edu/~zkincaid/courses/fall18/readings/SATHandbook-CDCL.pdf