SAT Assignment Part 2 – Non-Consecutive Sudoku SAT Solver

Andrzej Szczepura, Patrick Koopmann

04/11/2025

Overview

In this assignment you will build your own SAT solver and use it to decide whether given non-consecutive Sudoku puzzles are solvable. You are provided with three starter files:

- `main.py` – loads a puzzle, encodes it, calls your solver, and prints the result.
- `encoder.py` – empty - replace with your own implementation
- `solver.py` – where you will implement the actual SAT solver.

Puzzle Database

Along with this assignment, you are given a folder of puzzle files and a manifest.

- The puzzles come in three sizes: 9 × 9, 16 × 16, and 25 × 25.
- Puzzle files are plain text, each containing an $N \times N$ grid of integers ($0 =$ empty cell).
- The manifest lists the puzzles and their expected `SAT/UNSAT` status.
- Note: the SAT puzzles are not hand-designed "newspaper" Sudokus. They may have more than one valid solution. For this assignment, you only need to decide if a solution exists.

Your Task

Implement the solver in `solver.py`. The grader will run many puzzles through your code, so your solver must work correctly for all three sizes.

Solver Function

In `solver.py` you must implement:

```
def solve_cnf(clauses, num_vars):
    """
    clauses: list of clauses, each clause is a list of ints
    num_vars: total number of variables

    Return:
        ("SAT", model)  where model is a list of ints, or
        ("UNSAT", None)
    """
```

Return `"SAT"` if the formula is satisfiable, along with a model (any valid assignment), or `"UNSAT"` otherwise.

Running

To test your solver on a puzzle:

```
python main.py --in puzzle.txt
```

To test your solver on a set of clauses in DIMACS format:

```
 python main.py --in clauses.sat --sat
```

In both cases, your program must print **exactly one line**, either:

```
SAT
```

or

```
UNSAT
```

Competition Rules

All groups will be graded on the same set of puzzles. The competition is structured as follows:

- **Correctness** is the primary criterion. A solver that misclassifies any puzzle will be ranked lower than one that is correct on all puzzles.
- If multiple groups achieve full correctness, then **runtime** will be used as a tie-breaker. Faster solvers will be ranked higher.

Submission

Submit a ZIP archive named:

```
group_<number>_a2.zip
```

containing exactly:

- `main.py`
- `encoder.py`
- `solver.py`

Your submission must run with:

```
python main.py --in puzzle.txt
```

to solve the puzzle in puzzle.txt, and with

```
python main.py --in problem.sat --sat
```

to solve a given sat problem.

Deadline: 23:59 14-11-2025

Hints

- Begin with a simple DPLL-style solver to get correctness.
- Optimize later (unit propagation, branching heuristics, clause learning) for speed.
- Start testing with small 9 × 9 puzzles before moving up.