SAT Solvers for Sudoku Problems

Huber, Gregory V00862879
Hsu, Leo V00928098
Kashike Umemura V00909836

February 17, 2023

1 Introduction and Background

In this project, programs are written to fully solve partially solved Sudoku puzzles using the SAT solver, `miniSAT`. In particular, we have programs `sud2sat1` and `sat2sud1` that, respectively, reads a single Sudoku puzzle before converting it to a CNF formula which is then suited to be inputted to `miniSAT` to be solved, and translates the output produced by `miniSAT` for a given puzzle instance back to a solved Sudoku puzzle.

The general approach in which `sud2sat1` is implemented is that for each cell (i, j) in the puzzle taking on a possible value k we have a propositional variable x_{ijk}, and for each of x_{ijk} we create a unique integer for it given by

$$81 \times (i-1) + 9 \times (j-1) + (k-1) + 1 \tag{1}$$

where we associate ijk with a base-9 number with the final additional 1 for the encoding to be suited for `miniSAT`. With this encoding for the propositional variables, we start with a formula with the 'minimal' encoding for the constraints (page 5 of the CSC 322 lecture slides on SAT Solvers). Finally, we add to the formula singular terms (conjuncts with only one variable) for each known value from the given puzzle before we feed the formula (in DIMACS format) to `miniSAT`.

For `sat2sud1`, the approach is fairly straightforward. That is, we parse through the output of `miniSAT` (a satisfying assignment) to look for all the positive integer values (that correspond to the propositional variables that are assigned `TRUE`) and, in essence, recover the triple (i, j, k) from equation (1) and reconstruct the puzzle with cell (i, j) taking on value k.

The extended tasks are also done in this project, where we have programs `sud2sat2` (*efficient encoding*), `sud2sat2`, `sat2sud3` (*extended encoding*), and `sat2sud3`. Note that since for each $1 \leq i \leq 3$, `sat2sud`i simply translates a truth assignment of the propositional variables back to a solved puzzle configuration, it makes sense that their implementations coincide with one another, and it turns out to be the case for us this project. Moreover, we also observe that `sud2sat2` is obtained from `sud2sat1` by adding an additional constraint, and that `sud2sat3` is obtained from `sud2sat2` by adding 3 additional constraints.

2 Performance Evaluation

The statistics of every task were saved to a text file and analysed. The specific attributes that are analysed are `Number of clauses`, `Memory usage`, and `CPU time`.

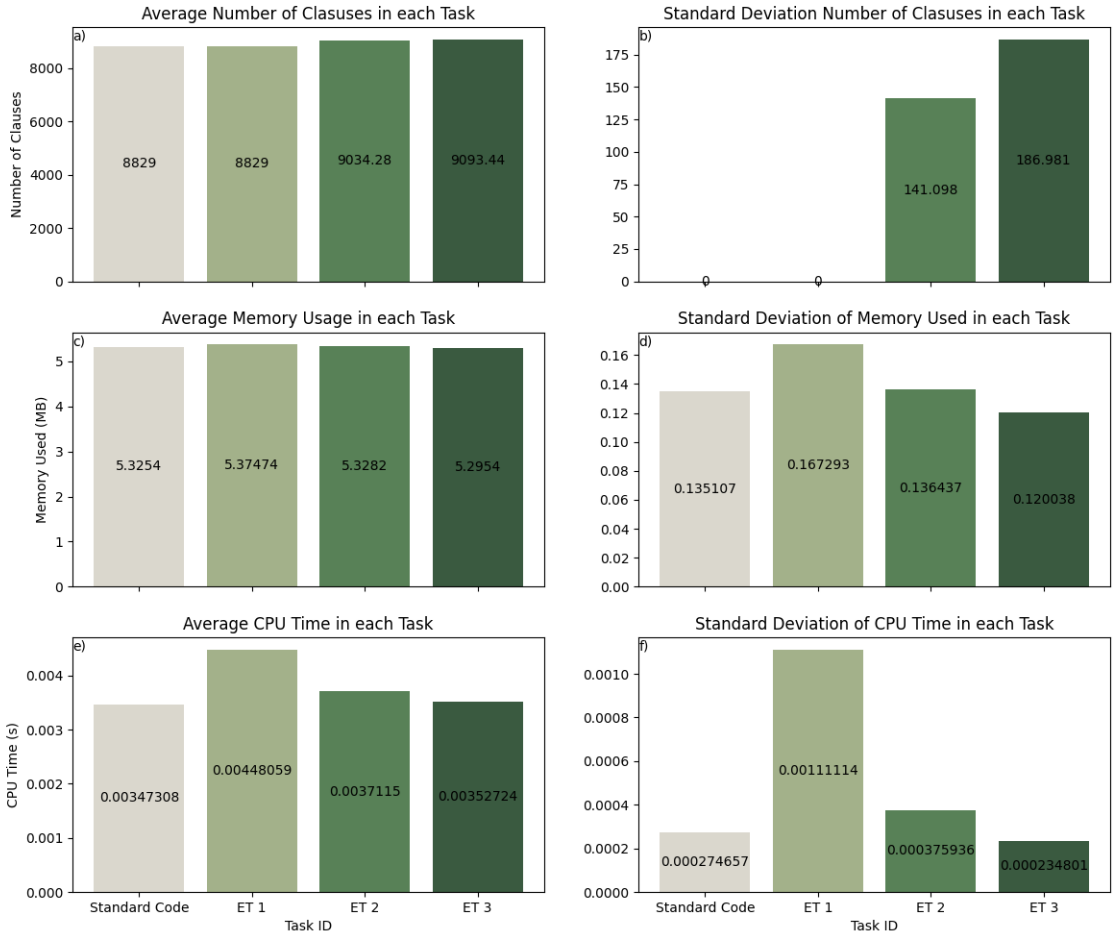

Figure 1: Average of the attributes `Number of Clauses`, `Memory Usage`, and `CPU Time` are shown on the charts a), c), and e) and the standard deviations of these attributes are shown on the charts b), d), and f) for each of the tasks which are labelled on the x-axis at the bottom of the figure. Standard Code is the original objective of this project, ET 1 represents the extended task 1, ET 2 the extended task 2, and ET 3 the extended task 3.

Figure 1 shows the averages and the standard deviation of each aforementioned attributes. In terms of averages, the `Number of clauses` and `Memory usage` the differences were insignificant for the two attributes. But for the `CPU time`, the extended task 2 took longer than any other task. This indicates that having more clauses does not necessary mean that it will take a longer `CPU time`. On the other hand, if we look at the standard deviations, for `Number of clauses`, there the minimum encoding and the extended task 1 has 0 as the standard deviation. This is because the standard code and the extended task 1 use the *minimal encoding*, so these encodings use the least number of clauses. However,

extended task 2 and 3 have some variance which are approximately 141 and 187 respectively. The standard deviation of `Memory used` were similar for the standard code, extended task 2 and extended task 3. Extended task 1 had a standard deviation value of approximately 0.03MB greater than the other values. As for the last attribute, `CPU time` the extended task 1 has the largest standard deviation, this would be expected, as the difference in the difficulty of the hard Sudokus are not likely to be similar, where ones with less pre-filled in cells are significantly harder.

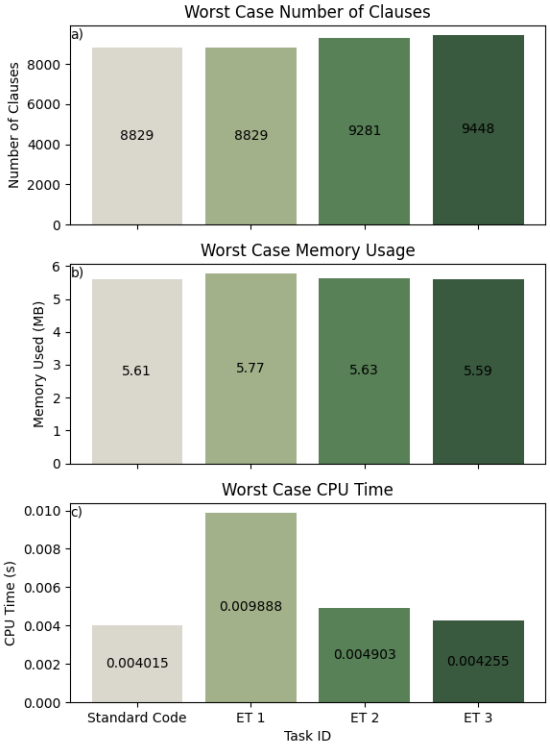

Figure 2: Attributes (`Number of clauses`, `Memory usage`, and `CPU time`) for each task in their worst results that were recorded by `minisat`. Just like *Figure 1* the tasks are labelled on the x-axis. a) represents the worst case number of clauses for each task, b) represents the worst case memory usage for each task and similarly c) shows the worst case CPU time for each task.

Figure 2 shows the worst case values of attributes `Number of clauses`, `Memory usage`, and `CPU time`. The task that has the most clauses is extended task 3. This is expected since this encoding is called the *extended encoding*, which uses all 4 of the additional constraints. The task with the second most clauses is the extended task 2, which includes the first additional constraints. The *minimal encoding* and the extended task 1 have the same number of clauses because they both use the same constraints, which is minimal. The `Memory usage` of all tasks were similar for the worst case. But on the other hand, the `CPU time` is significantly worse for extended task 1. This supports the possibility that having more clauses does not increase computation time but having a difficult Sudoku puzzle does increase it.