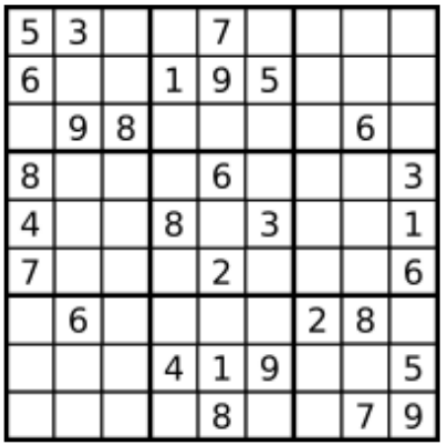
Sudoku SAT Solver



Elif Hangül
Hamit Kavas
Jonatan Koren

Feb 2020
Autonomous Systems

Exercise A.1

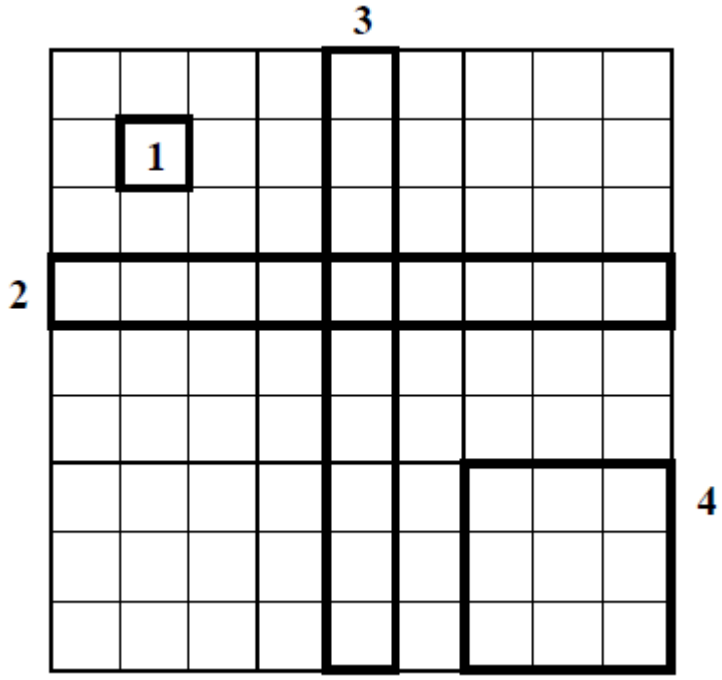
Since Sudoku problem's itself has been defined as Constraint Satisfaction problem[1], determining of constraints has vital importance in this project. So, we have neatly tried to select the most useful and minimal encodings.

The goal in this game is to fill any blank entries with numbers from 1 to 9 such that every row, every column and every 3x3 sub-grid contains each of nine possible numbers.
Fun fact; Sudoku means "single number" in Japanese.

Here we will represent sudoku as SAT problem where we will use n propositional variables which can be assigned truth values 0 (false) or 1 (true). A literal that is assigned to truth value 1 satisfies the clause, and the clause is said to be satisfied. The formula is satisfied if all its clauses are satisfied.

a)

Encoding Sudoku puzzles into CNF requires 9.9.9 = 729 propositional variables. For each entry in the 9x9 grid, we associate 9 variables.[2] We will have 4 groups of constraints as it has been shown on puzzle below.



Let us use s_{xyz} to refer to variables where it is assigned true *if and only if* the entry in row *x* and column *y* is assigned to number *z*.
Constraint 1) ***Each cell must contain a number*****:** "There is at least one number in each entry"

$$\bigwedge_{x=1}^{9} \bigwedge_{y=1}^{9} \bigvee_{z=1}^{9} s_{xyz}$$

Constraint 2) ***Each number appears at most once in each row:***

$$\bigwedge_{y=1}^{9}\bigwedge_{z=1}^{9}\bigwedge_{x=1}^{8}\bigvee_{i=x+1}^{9}(\neg S_{xyz} \vee \neg S_{iyz})$$

Constraint 3) ***Each number appears at most once in each column:***

$$\bigwedge_{y=1}^{9}\bigwedge_{z=1}^{9}\bigwedge_{y=1}^{8}\bigvee_{i=y+1}^{9}(\neg S_{xyz} \vee \neg S_{xiz})$$

Constraint 4a) ***Each number appears at most once in each 3X3 sub-grid row:***

$$\bigwedge_{z=1}^{9}\bigwedge_{i=0}^{2}\bigwedge_{j=0}^{2}\bigwedge_{x=1}^{3}\bigwedge_{y=1}^{3}\bigwedge_{k=y+1}^{3}(\neg S_{(3i+x)(3j+y)z} \vee \neg S_{(3i+x)(3j+k)\,z})$$

Constraint 4b) ***Each number appears at most once in each 3X3 sub-grid column:***

$$\bigwedge_{z=1}^{9}\bigwedge_{i=0}^{2}\bigwedge_{j=0}^{2}\bigwedge_{x=1}^{3}\bigwedge_{y=1}^{3}\bigwedge_{k=x+1}^{3}\bigwedge_{l=1}^{3}(\neg S_{(3i+x)(3j+y)z} \vee \neg S_{(3i+k)(3j+l)\,z})$$

b)

Code has been provided in sudoku.py. Example output is shown below. The resulting CNF formula has 8846 clauses, including the unit clauses representing the pre-assigned entries. From these clauses, 81 clauses are nine-ary and the remaining clauses are binary clauses.

```
D:\Jupyter\Autonomous Systems\sat\lab2>python sudoku.py .................1.....2.3......3.2...1.4......5....6..3......4.7..8...962...7...
Writing SAT problem with 729 vars and 8846 clauses to file "theory.cnf"
================================[MINISAT]================================
| Conflicts |     ORIGINAL     |              LEARNT              | Progress |
|           | Clauses Literals |   Limit Clauses Literals  Lit/Cl |          |
=========================================================================
|         0 |    3278    11170 |    1092       0        0    -nan |  0.000 % |
=========================================================================
restarts              : 1
conflicts             : 16             (348 /sec)
decisions             : 63             (1370 /sec)
propagations          : 1872           (40696 /sec)
conflict literals     : 127            (0.00 % deleted)
Memory used           : 5.56 MB
CPU time              : 0.046 s

SATISFIABLE
Solution: 9531687428627349514179528367468931252816453973952714681385296745743862196294175 83
Solution in board form:
9 5 3 | 1 6 8 | 7 4 2
8 6 2 | 7 3 4 | 9 5 1
4 1 7 | 9 5 2 | 8 3 6

7 4 6 | 8 9 3 | 1 2 5
2 8 1 | 6 4 5 | 3 9 7
3 9 5 | 2 7 1 | 4 6 8

1 3 8 | 5 2 9 | 6 7 4
5 7 4 | 3 8 6 | 2 1 9
6 2 9 | 4 1 7 | 5 8 3
```

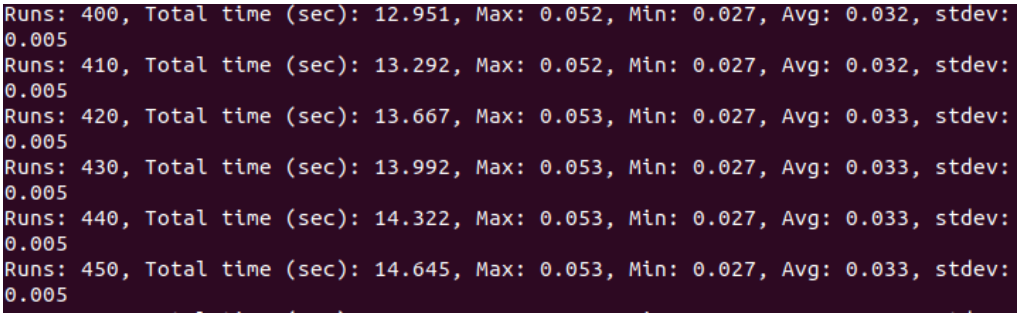
c)

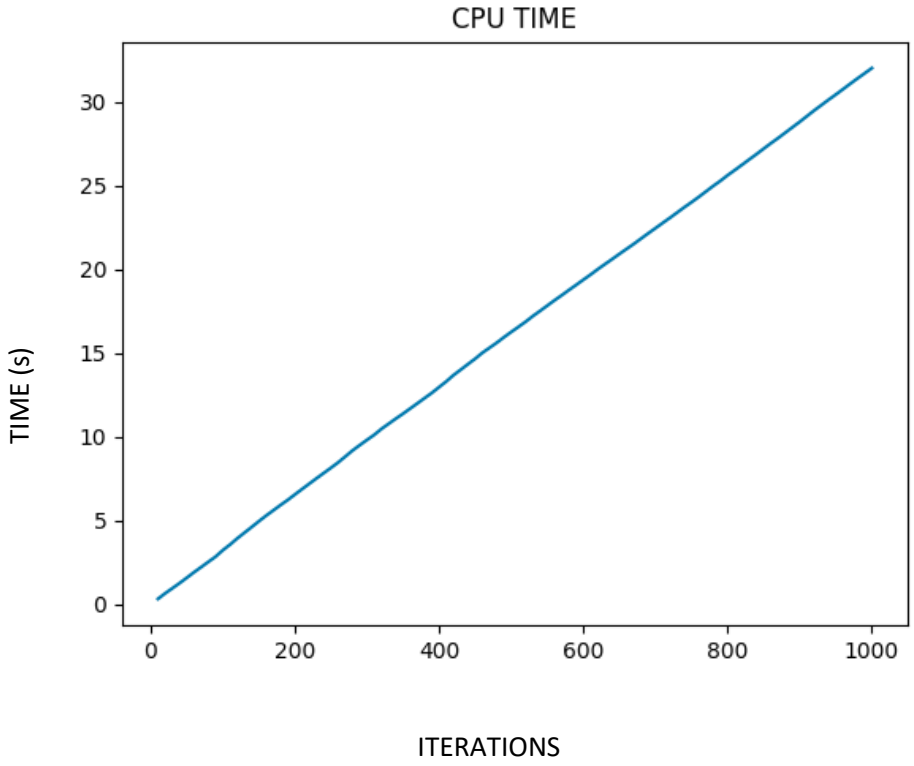
In order to count the number of possible solutions we are considering every possible value a square in sudoku board could have. For each box could be filled with a number between 1 and 9 except the predefined entries. For each variation, a new board combination is created and find_one_solution method is called to see if a viable option is found. If there is such option, counter is incremented by one. At the end counter holds the number of all possible solutions.

The code is tried with puzzles from puzzles1_17_clue file inside of benchmarks directory. Usually the puzzles we tried had 1 solution. Then we continued by deleting the predefined entries one by one, count the number of all possible solutions for the new board and we saw an increase in the numbers. Removing even one entry could increase the number of solutions to three hundreds. We could remove 8 entries for the first puzzle in working file after the code started to have more than one minute of compilation time.

d)

The number of the variables is 729 and the number of clauses is 8,846.
As the number of n increases, the number of variables and clauses respectively.
For example, when running this experiment for 1000, we observe that the average time for each calculation is pretty much the same (~ 0.032 seconds) and thus the total amount of time is increasing linearly.
The maximum CPU time for each puzzle is on average 2 times the minimum CPU time (~0.027 / ~0.054 seconds).
Generally, as the number increases, the average CPU time increases a little bit as well.
The standard deviation is also stable for each puzzle (~ 0.05 seconds).
example of running a large number of instances from the “puzzles1_17_clue” (from benchmark folder).





References

1- Simonis, Helmut. (2020). Sudoku as a constraint problem.
2- Ivor Spence, Available at: *http://www.cs.qub.ac.uk/~I.Spence/SuDoku/SuDoku.html* (Accessed: 20.02.2020).
3- Lynce, Inês & Ouaknine, Joël. (2006). Sudoku as a SAT problem.