INFO9015-1 : Logic for Computer Science

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

Faculty of Applied Science

Sudoku solving using the SAT4J solver

Teacher :
Pascal FONTAINE

Assistant :
Tanja SCHINDLER

Student :
Arthur LOUIS, s191230

November 29, 2022

Contents

List of Figures

List of Tables

1 Encoding to SAT

1.1 Solving sudokus

To solve the sudokus, one can encode the problem as a SAT problem and use a SAT solver like SAT4J to solve it. For this project, the choosen encoding for the element at the line i and column j of a size-N sudoku with the value k (filled with leading zeros to have length equal to 2) is the following:

$$\text{variable} = (i \times N + j) \times 100 + k$$

To create the clauses, two functions were implemented: `newlit(i,j,k,N)` and `newneglit(i,j,k,N)`. The first one creates a new literal with the value of the variable described above and the second one creates a new literal with the negation of the variable described above.

To solve the sudoku, 8 rules were implemented by looping in a correct way to create the clauses:

- Each cell contains at least number between 1 and N.
- Each cell contains at most one number between 1 and N.
- Each line contains every number once.
- For each line, every number appears at most once.
- Each column contains every number once.
- For each column, every number appears at most once.
- Each block contains every number once.
- For each block, every number appears at most once.

1.2 Checking sudokus uniqueness

Once that one can solve a sudoku with the SAT solver, it is straightforward to check if the sudoku has a unique solution. Indeed, simply giving back the same input to the SAT solver won't do the trick as it is deterministic. One can simply add a line to the clauses that forbids the solution found by the SAT solver to be the solution of the sudoku. If the SAT solver finds another solution, the the sudoku is not unique. If the SAT solver doesn't find any solution, the sudoku is unique. This technique is simply implemented in the function `sudoku_other_solution_constraints` by looping on the solution found by the SAT solver with `newneglit`.

1.3 Creating sudokus

1.3.1 Creation with all numbers

To create the sudoku, one can simply begin by a blank sudoku and creating some kind of random seed for the SAT solver by placing randomly each number once in the sudoku, thus leading to no incoherence. From that random seed, one can use the SAT solver to find a solution and from that solution, one can remove a number at a time by checking everytime that the sudoku is still solvable and unique. If not, the removed number is reintroduced in the sudoku. This technique was implemented in the function `sudoku_generate`. Once the function finds a sudoku that is solvable and unique, it returns it and saves it in a file called `generated_sudoku.txt`.

1.3.2 Creation with only 8 numbers

To create a sudoku with only 8 numbers, one can simply implement the technique formulated above but before the removing phase of the protocol begins, all the clues for a certain value are removed. Then the looping starts and as previously, once the function finds a solvable and unique sudoku, it returns it and saves it in a file called `generated_sudoku.txt`. This technique was implemented in the function `sudoku_generate` and by passing the value `True` to the argument `cm`.

2 Results obtained by the solver

The results of my implementation of the sudoku solver are presented in the table below. The commputation was done on a MacBook Pro 2020 with a 1.4 GHz Quad-Core Intel Core i5 and 8 Go of 2133 MHz LPDDR3 RAM.

Size	9x9	16x16	25x25
Mean	0.1834s	0.4765s	2.0894s
Std	0.0054s	0.0170s	0.0814s
Max	0.1977s	0.5182s	2.2260s
Min	0.1610s	0.4569s	2.0260s

Table 1: Solving times of the SAT solver

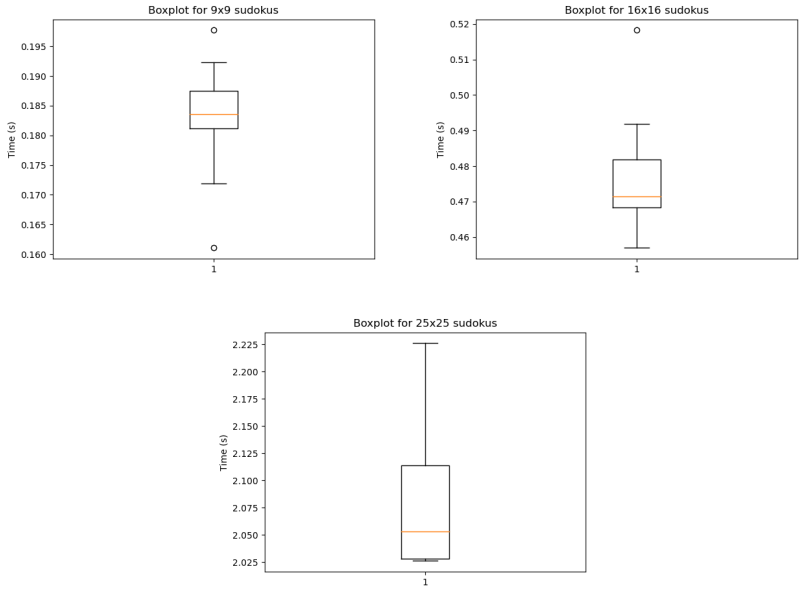

Figure 1: Boxplot of the solving times of the SAT solver