

SAT/SMT Solving

REPORT

Sudoku solver and generator

PRIVITERA Concetto Antonino
MOSIG M2

Academic year 2021-2022

Contents

1	Introduction	3
2	Files and requirements	3
3	Sudoku solver	5
3.1	Z3 Library	5
3.2	Backtracking	6
4	Sudoku generation	6
5	Conclusion	9

1 Introduction

The goal of this report is to give a possible application to solve and generate a Sudoku. In particular, there are two proposed solutions to solve it: through the Z3 library and backtracking algorithm.

In the next sections, it is described the files related to this report, the requirements needed, how to use the application and how it works.

2 Files and requirements

The project, made in python and commented in some parts, contains the following files:

```
├── sudoku_solver.py
├── sudoku_lib.py
└── data
    ├── 4x4.txt
    ├── 9x9.txt
    ├── 16x16.txt
    └── 25x25.txt
```

The file *sudoku_lib.py* contains the all functions to read a file containing a Sudoku, solve or generate one and also some functions to check the correctness of Sudoku structure and grid size. Indeed, there are some functions to check that the grid size defined in the file is accepted and also its grid data structure is correct.

There are also files contained within the folder *data* which are some samples of Sudoku to solve through the application. Here is an example of the data structure contained in the file *4x4.txt*:

```
4
- - - -
1 2 4 -
2 - - -
- 4 1 -
```

Figure 1: *4x4.txt* content

As it is possible to notice, the first row represent the grid size number, while the other rows are actually part of the Sudoku grid. Indeed, it is composed by numbers or by empty fields represented by `-` symbol. Also, once the grid is read, it is stored into the memory as

matrix containing numbers or 0 to identify empty fields.

Instead, the file *sudoku_solver.py* is a sort of terminal able to use the functions of *sudoku-lib.py*.

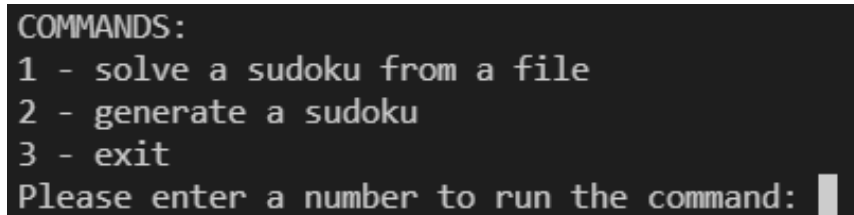
In order to run the application, it is needed **python 3** and the **Z3 library** which can be installed through the terminal by running the command:

```
1 pip install z3-solver
```

After that, the application can be run through the file *sudoku_solver.py* on any editor such as VS code or by running:

```
1 python sudoku_solver.py
```

Once the application is running, you get the following interface:



```
COMMANDS:  
1 - solve a sudoku from a file  
2 - generate a sudoku  
3 - exit  
Please enter a number to run the command: █
```

Figure 2: Interface commands

It is just needed to choose a command by typing a number and following the instructions on screen.

3 Sudoku solver

After the application is launched, if you type *1*, then you can solve a Sudoku stored in any file that contains the correct Sudoku structure showed before. Also, it is also necessary give the filename and the type of solver which can be Z3 or backtracking. In figure 3 it is shown an example of usage.

```

COMMANDS:
1 - solve a sudoku from a file
2 - generate a sudoku
3 - exit
Please enter a number to run the command: 1
You selected 1 to solve a sudoku from a file
Please add the file name which contains your sudoku (empty if you want to use data/9x9.txt as default): data/4x4.txt
your file contains the following sudoku to solve:
- - - -
1 2 4 -
- - - -
2 - - -
- 4 1 -

Which solver do you want to use? (1 - z3, 2 - backtracking): 1
4 3 2 1
1 2 4 3

2 1 3 4
3 4 1 2

--- computation time: 0.03889942169189453 seconds ---

```

Figure 3: Sudoku solver example of a Sudoku 4x4

The application prints the original Sudoku as feedback after reading the file and the solution after choosing the type of solver.

3.1 Z3 Library

A first way to solve a Sudoku is by using the Z3 library, thanks to this prover it is possible to solve any Sudoku with the grid size defined. However, in order to work correctly, it is necessary to defines the whole constraints to make the problem solvable. The matrix has to satisfy the following constraints:

- Each cell has to contain a number between *1* and *N* for a grid $N \times N$;
- Each row has to contain unique numbers;
- Each column has to contain unique numbers;
- Each sub-grid has to contain unique numbers;
- The numbers contained in the Sudoku problem have to be kept in the solution.

Once the the solver is executed, it returns a solution if found which contains a number for every cell of the grid. The time needed to solve is typically low which takes few seconds for the bigger grid 25×25 .

3.2 Backtracking

The other way to solve a Sudoku is through an algorithm which uses the technique of backtracking. This is possible thanks to the recursive approach to have a backup during the execution to cancel the steps done if needed. At the same way, this algorithm uses the same constraints of the Z3 library, but with a different method.

It works in this way:

1. It starts from the first cell of the grid until it reaches the last one;
2. If the cell already contains a number defined by the problem, then skip it;
3. If the cell is empty, then try a possible value between 1 and N for a grid $N \times N$;
4. If the value chosen satisfies the constraints, then read then next cell and repeat from step 2 on until the last cell is reached to found a solution;
5. In case no value was found that satisfies the constraints, then go back to the previous cell, try a new value and cancel the last change;
6. If no value was found for the first cell, then the Sudoku does not accept any solution.

Although this method is still a valid Sudoku solver, it is slower than the z3 implementation as it is possible to infer. Indeed, this can be notice more when trying to solver bigger grid as 25×25 .

4 Sudoku generation

This section is more focused on the Sudoku generation and it gives some details on the internal mechanism and the techniques used to satisfy some particular constraints.

The generation process is composed by three main important steps in order to make it fast enough:

1. Generate the independent sub-grids
2. Use a solver to generate the remaining cells
3. Remove the cells and keep the uniqueness of the solution

The first step is particularly important because it generally helps to speed up the whole process. Its goal is to generate the independent sub-grids. In other words, it randomly generates a sub-grid and place it in the main grid where no other sub-grids are in the same rows and columns. In this way, it is not needed to check the Sudoku constraints for this step anymore since they are mathematically kept. This means, the first step just generates, for example, the diagonal or others that are shown in the figure in the next figure.

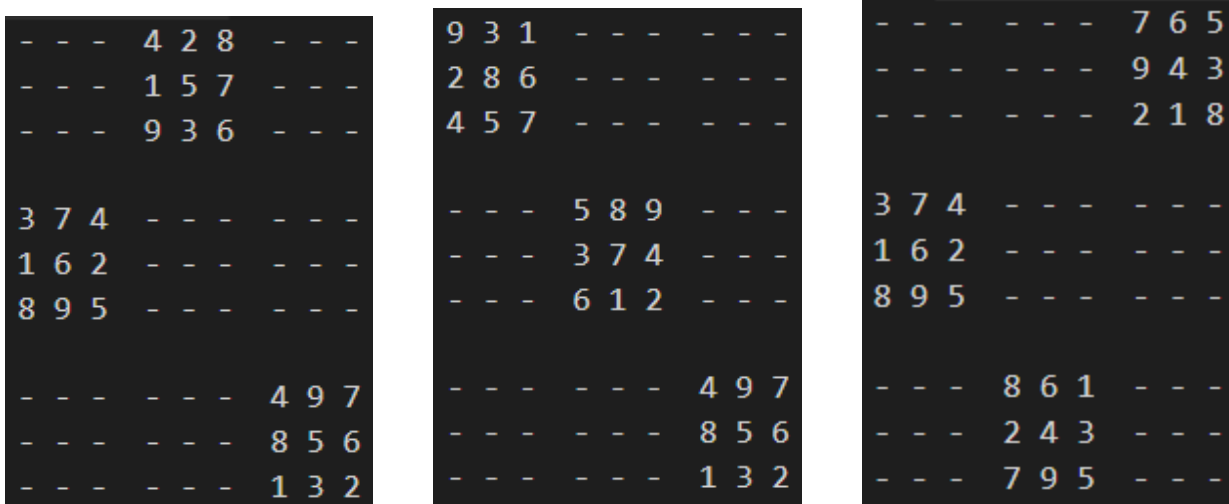


Figure 4: Some possible generations of the first step for a grid 9×9

The second step consists in generate the remaining cells by using a solver. Here the application allows you to use Z3 library or the backtracking algorithm in which both have some advantages and disadvantages.

If you want to use Z3 library, then it is much faster, but its generation is deterministic once the first step is over. However, thanks to the first step mechanism, we still keep the generation randomness. Instead, the backtracking is slower, but add further randomness to generate the remaining cells and find the final Sudoku solution.

The third and last step is the one that actually generates the final grid. Its mechanism is composed by the following operations:

1. It starts from the Sudoku solution obtained by the previous step
2. Through an iterative approach composed by N attempts, it chooses a random cell not yet analyzed and try to remove
3. After removing a cell, check if the Sudoku it's keeping the uniqueness solution by find a new different possible solution by using Z3 to speed up the execution
4. If the uniqueness is kept, then save the grid without that cell and repeat from operation 2 by using this last grid
5. If the uniqueness is violated, then restore the cell removed, reduce the number of attempts and repeat operation 2
6. Once every cell is tested or the number of attempts is 0, then return the all possible grids found and let the user to choose how many cells remove to obtain the final Sudoku

The number of attempts in operation 2 is defined by the user and it is just a way to speed up more the execution. The idea is that the more cells are removed the harder is to keep the solution uniqueness. Each attempt corresponds to a tested cell. So, it is possible try every cell by setting a higher number of attempts or stop it immediately if the number of attempts is 0.

The operation 3 is using Z3 to check the uniqueness. Indeed, thanks to it, the generation time ranges from few seconds to few minutes for the bigger grid 25×25 . Its goal is to make sure that every empty cell cannot contain a value different from the original solution and still being solvable. If a different value is found, then it is not possible to remove that cell because otherwise the Sudoku would have multiple solution.

Regarding operation 6, every time a cell is removed correctly, then the grid with once cell less is updated and stored incrementally into an array of grids. This let the user choose a grid with a specific number of cells removed which is also synonym of difficulty.

Here is an example to generate a Sudoku 4×4 step by step showed in figure 5.

```

COMMANDS:
1 - solve a sudoku from a file
2 - generate a sudoku
3 - exit
Please enter a number to run the command: 2
You selected 2 to generate a sudoku
Choose a solver to fill your sudoku
Do you want to use z3 (1 - default) or backtracking (2)? 1
Grid size available:
4 - 4x4
9 - 9x9
16 - 16x16
25 - 25x25
Choose the grid size:4
How many attempts do you want to try to remove a new cell and keep the unicity solution?
Type a number between 0 and on (5 is default): 5
Sudoku with grid size 4
Your sudoku solved is:
2 1 4 3
4 3 2 1

1 4 3 2
3 2 1 4

Removing cells with 5 attempts...
--- computation time: 0.568911075592041 seconds ---
Number of cells removed are 12 after 5 attempts.
How many cells do you want to remove between 1 and 12? 12
Your final sudoku is:
- - - -
4 - - 1
1 - - -
- 2 - -

Your final sudoku in file format is:
4
- - - -
4 - - 1
1 - - -
- 2 - -

```

Figure 5: Sudoku generation example of a Sudoku 4×4

As it is possible to see, the application also prints the final Sudoku in file format in order to copy it in a file and solve it through the proposed solvers.

5 Conclusion

The application made in Python allows the user to set and test different options for both Sudoku solver and generation. Different techniques are proposed with their benefits and drawbacks.

In particular, Z3 library and backtracking algorithm are used to solve or to generate a matrix. Generally, the Z3 library turns out better than the backtracking method. However, the latter it is a still valid solution that can be used as well.

In conclusion, this tool allows, not only to solve a Sudoku or to make a new one, but also to experiment and compare a prover and a backtracking algorithm in terms of speed and results.