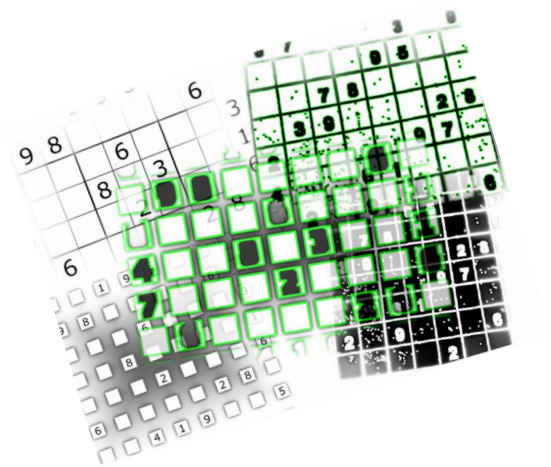
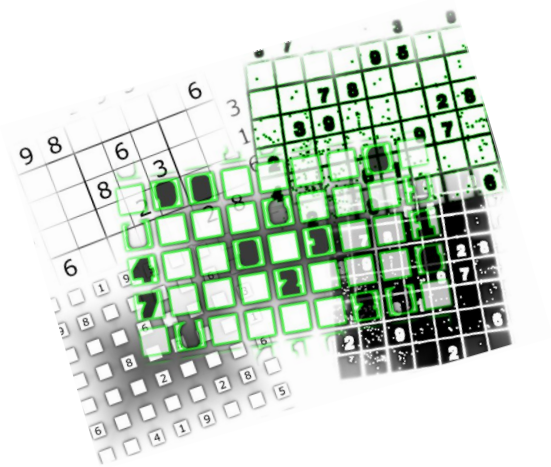


Sudoku and Optical Character Recognition in Python

IDA Østjylland Seminar, 23/10/2023

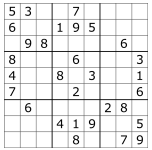
Luminita C. Totu
Control Engineer
luminita.totu@gmail.com



Sudoku ~~and~~ ~~Optical Character Recognition~~ in Python

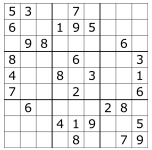
IDA Østjylland Seminar, 23/10/2023

Luminita C. Totu
Control Engineer
luminita.totu@gmail.com



Sudoku - What is It ?

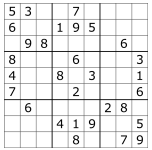
- https://www.sudopedia.org/
- Grid, Cells, Houses
- Strategies for human solving: Singles, Pencilmarks, Scanning, Cross-Hatching, Bifurcation, Snyder Notation,...
- Variations: e.g. Killer sudoku with arithmetic cages
- https://www.youtube.com/c/CrackingTheCryptic (also a few podcast episodes)
- https://hodoku.sourceforge.net/en/techniques.php



Sudoku: A Classical Solving Algorithm

- Overview
- Data Structures
- Functions
- Main methods

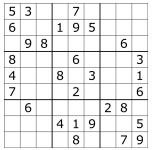
Fjends folkeblade, 2020



Sudoku: A Classical Solving Algorithm

- Calculate a list of all possible choices for each free/empty cell
- Where there is only one choice available fill in the value in, AND perform the necessary elimination from the previous lists of all possible choices
- When there no empty cells with only one choice left, looks for an empty square with the fewest possible choices, and pick the first (or a random) value from the list
- If the Sudoku becomes blocked, backtrack to the last “bifurcation point” and take the next option …

// For a slightly different approach see also this https://norvig.com/sudoku.html

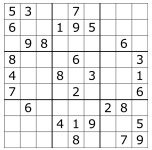


Sudoku: A Solving Algorithm: Data Structures

- using two main data storage arrays:
  - a 2-dimensional (2d),
  - a 3-dimensional (3d)

```
sudoku_1 = np.array([
                [ 0, 0, 0, 0, 0, 0, 0, 0, 0 ],  # nul betyder tom plads
                [ 4, 0, 9, 7, 0, 2, 1, 0, 3 ],
                [ 2, 0, 3, 4, 0, 9, 7, 0, 6 ],
                [ 0, 4, 0, 0, 0, 0, 0, 9, 0 ],
                [ 8, 0, 2, 0, 0, 0, 4, 0, 5 ],
                [ 0, 3, 0, 0, 0, 0, 0, 1, 0 ],
                [ 1, 0, 7, 3, 0, 4, 5, 0, 8 ],
                [ 6, 0, 8, 2, 0, 1, 3, 0, 9 ],
                [ 0, 0, 0, 0, 0, 0, 0, 0, 0 ] ] )
```

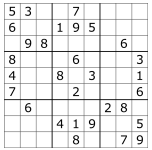
- the 2d data storage is the matrix of the Sudoku; i contains the known/fixed numbers, and gradually, as the algorithm works its way, it becomes more and more full. An empty place is denoted by **zero**.



Sudoku: A Solving Algorithm: Data Structures

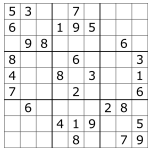
- The 3d data storage is to keep track of all possible choices for the Sudoku, for the empty places. For each (i,j) coordinate of the Sudoku matrix, we have a new dimension of size/length 9.
- In Python/NumPy, the coordinates of arrays start at zero and go to n-1, where n is the size of the array in the given dimension.
- Zero is used in the possibility matrix for padding, in order to have an array of consistent/fixed size.

```
sudoku_possibilities = np.zeros([9,9,9])   # sudoku_possibilities

...

...
print(sudoku_possibilities_test[0,0,:]) # [ 5.  7.  0.  0. 0. 0. 0. 0. 0.]
print(sudoku_possibilities_test[0,1,:]) # [ 1.  5.  6.  7. 8. 0. 0. 0. 0.]
print(sudoku_possibilities_test[8,8,:]) # [ 1.  2.  4.  7. 0. 0. 0. 0. 0.]
```



Sudoku: A Solving Algorithm: Functions

- **is_number_k_in_line_i**(k,i,sudoku_fixed) // **er_der_tal_i_linjen**
- **is_number_k_in_column_j**(k,j,sudoku_fixed) // **er_der_tal_i_soejlen**
- **is_number_k_in_square_corresponding_to_i_j**(k,i,j,sudoku_fixed) // er_der_tal_i_kvadraten
- **can_number_k_be_here_at_i_j**(k,i,j,sudoku_fixed) // **kan_k_vaere_her**
    - Combination of the three previous functions
- **what_can_be_here_at_i_j**(i,j,sudoku_fixed): // **hvad_kan_vaere_her**
    - A for loop over the previous function

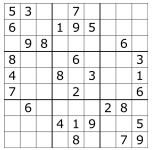


Sudoku: A Solving Algorithm: Functions

- **fill_possibilities** // **skab_muligheder**

```
Sudoku_possibilities = np.zeros([9,9,9])
for i in np.arange(0,9):
    for j in np.arange(0,9):
        sudoku_possibilities[i,j,:] = what_can_be_here_at_i_j(i,j,sudoku_fixed)
```

- **remove_possibilities**(k,i,j,sudoku_possibilities) // **fjern_muligheder**
  - remove number k put at line i, from all columns
  - remove number k put at column j, from all lines
  - remove k put at (i,j) from the associated square



Sudoku: A Solving Algorithm: Functions

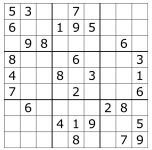
- **is_sudoku_blocked**

  A blocked situation is when at least one free space has an empty possibility list

  (Because we might need to choose at random and backtrack)

- **is_sudoku_finished**

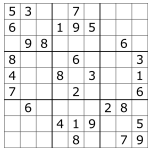
  No more empty(0) cells



Sudoku: A Solving Algorithm: Main Methods

method_sub_A, method_A:

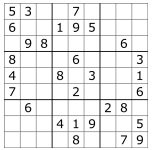
- Identify locations in the Sudoku free cells where there is only one possibility, fill it in, update the possibility matrix and repeat / try again !
- Until either:
  - **Sudoku is solved (1)**
  - **No places with just one possibility are found (2)**
  - **Sudoku becomes blocked (3)** (and this is because of method_B …)



Sudoku: A Solving Algorithm: Main Methods

method_B:

- when there are no free cells with only one digit possible
- an approach is to look for places with the fewest choices (this is what a human player would probably do as well), and choose/guess one of the options
- in this implementation, the choice/guess in made in an ordered manner, the smallest numbers first
- So what next ?

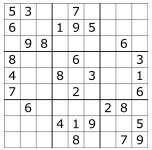


Sudoku: A Solving Algorithm: Main Methods

method_B:

- after choosing/guessing a number, method A is taken again until:
  - Sudoku is solved (1)
  - No places with just one possibility are found (2) → **Choose/Guess again**
  - Sudoku is blocked (3) → **We backtrack to the latest Choice/Guess**

- Recursion and the function call stack makes it possible, almost in a hidden way, to return/backtrack when the Sudoku is blocked. It’s like magic, we’ll look later in the code !



Sudoku: A Solving Algorithm

- Wikipedia's hard brute force Sudoku: https://en.wikipedia.org/wiki/File:Sudoku_puzzle_hard_for_brute_force.jpg

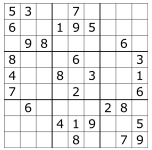
					3		8	5
		1		2				
			5		7			
		4				1		
	9							
5							7	3
		2		1				
				4				9

```
# MAIN ALGORITHME

sudoku_fast = np.copy(sudoku_9)
print(sudoku_fast)

# for alle tom kasse, finde mulighederene
sudoku_muligheder = skab_muligheder(sudoku_fast)

[ sudoku_klar, sudoku_fast, sudoku_muligheder ] = method_B(sudoku_fast,sudoku_muligheder)
```

✓ 2m 1.5s

[i=1,j=1] muligheder = [2. 3. 4. 6. 7. 8. 9.]
[i=1,j=2] muligheder = [2. 3. 4. 5. 6. 7. 8.]
[i=1,j=3] muligheder = [3. 5. 6. 7. 8. 9.]
[i=1,j=4] muligheder = [1. 4. 6. 7. 8. 9.]
[i=1,j=5] muligheder = [5. 6. 7. 8. 9.]
….

- Lowest list of choices was 3 (?) in the initial stage



Sudoku: Solving a Sudoku in Jupyter Notebook

Activity !