Aim: Investigating and comparing different stochastic sudoku solving algorithms

Create a test bed that takes a range of sudoku solving algorithms and compares them using a range of difficulty of puzzles.

Objectives:

Explore current methods of sudoku solving

Develop test software to run solvers through

Implement sudoku solving methods ranging from simplistic to complex

Compare implementations with one another

Explore current methods of sudoku solving

Establish test data for comparison of algorithms

Select sudoku solving algorithms to implement

Create test bed to allow comparison of algorithms

Evaluate implemented sudoku algorithms at multiple complexities

Backtracking: brute-force algorithm, fills cells in order putting a number starting at 1 and increasing, when a sudoku rule is broken by one of the numbers: backtracks to previous number in order and increases it until it complies with the rules. Keeps doing this working through the sudoku cell by cell until it finds solution.

<https://www.geeksforgeeks.org/sudoku-backtracking-7/>

<https://see.stanford.edu/materials/icspacs106b/H19-RecBacktrackExamples.pdf> **Page 8**

Solution always found from valid sudoku, however slowest possible method

Stochastic search: randomly assign numbers to each empty cell, calculate number of errors?, shuffle inserted numbers until number of errors is zero: **different methods for shuffling numbers.**

<https://arxiv.org/ftp/arxiv/papers/0805/0805.0697.pdf> 4 different approaches used

<https://pdfs.semanticscholar.org/2596/8256c1dd61c5c5ff13b3ecc2fb146259f171.pdf>

Modelling a sudoku as a constraint satisfaction problem: <https://en.wikipedia.org/wiki/Constraint_satisfaction_problem>

<https://en.wikipedia.org/wiki/Sudoku_solving_algorithms>

<http://orca.cf.ac.uk/27746/1/LEWIS%20metaheuristics%20can%20solve%20sudoku%20puzzles.pdf>

Inserts numbers 1-9 in each 3x3 box – solves constraint 3. Finds how many numbers are missing from each row and column and adds then up to a “cost”. Then swaps 2 numbers are re-evaluates the rows and columns that have changed. Repeats until the cost is 0.

algorithms that are used to solve sudoku can also be used to create valid complete sudoku using a blank grid and running the algorithm. It being “good” is dependent on the algorithm used, e.g. backtrack will make valid solution but will be “123456789” in order for each row.

<https://github.com/sraaphorst/sudoku_stochastic>

implementations of these algorithms in c++:

1. Genetic algorithm
2. Hill climbing
3. β-Hill climbing
4. Great deluge algorithm
5. Tabu search algorithm
6. Simulated annealing algorithm

Can get sudoku from here: <https://www.sudokuoftheday.com/dailypuzzles/>

<https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3>

<https://link.springer.com/content/pdf/10.1007%2F978-3-642-13498-2_60.pdf>

<https://www.adrian.idv.hk/2019-01-30-simanneal/>

<https://www.101computing.net/sudoku-generator-algorithm/>

IDEA FOR SIMPLE GUI: have first screen of tick boxes for algorithms that are to be tested and difficulties of board that they are to be tested with. 2 columns of tick boxes.

Then when it runs it opens up another window that displays the raw data of each algorithm and the comparison between them. Possibly have a graph but **look into that**.

Puzzle generation: <https://qqwing.com/>