Soduku Backtracking Algorithm Report

Aims:

Perform empirical analysis and compare the observation to the theoretical analysis of the Backtracking algorithm for solving 9×9 sudoku puzzles.

Summary of Theory:

Sudoku is a logic-based, combinatorial number-placement puzzle. In classic sudoku, the objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 subgrids that compose the grid (also called "boxes", "blocks", or "regions") contain all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a single solution.

A backtracking algorithm for soduku is a recursive algorithm that attempts to solve a given soduku puzzle by testing all possible configurations towards a solution until a solution is found. Each time a configuration is tested, if a solution is not found, the algorithm backtracks to test another possible configuration and so on till a solution is found or all configurations have been tested.

Time complexity: O(9^(n*n)).
For every unassigned index, there are 9 possible options so the time complexity is O(9^(n*n)). The time complexity remains the same but there will be some early pruning so the time taken will be much less than the naive algorithm but the upper bound time complexity remains the same.

Experimental Methodology:

Using our java implemented sudoku backtracking algorithm we will alter the algorithm to have a timer around where the problem will be solved. This timer will indicate the time it took to solve the sudoku puzzle. There will be different sudoku puzzle inputs, these puzzles will differ in the level of difficulty and all there time to completion will be recorded. We will further alter the code to have a counter to count the number of operations on the different puzzles. All of the outputs will be evaluated to come to a decisive conclusion.

Presentation of results:

Easy Inputs	Time	Operations
0 0 3 0 0 4 0 1 6 0 7 4 9 0 0 0 0 0 0 0 8 5 3 0 0 4 0 0 6 2 0 0 0 4 0 0 0 0 0 0 0 2 3 6 8 4 0 0 8 1 0 0 0 0 0 0 0 6 0 8 0 0 7 7 2 0 0 9 0 0 0 0 5 8 0 7 0 0 6 9 0	712796 nanoseconds = 0.7 milliseconds	82 operations
3 7 0 9 0 0 2 8 0 0 0 1 2 5 0 0 0 0 0 0 0 0 0 0 9 0 0 5 8 2 0 4 0 7 0 0 0 0 0 0 8 0 5 9 0 0 0 0 7 0 5 0 4 0 0 0 6 0 0 2 0 7 3 4 0 3 1 6 7 8 0 0 0 5 0 0 0 0 0 6 0	644607 nanoseconds = 0.6 milliseconds	82 operations
0 2 0 8 0 0 5 0 1 1 9 0 5 0 7 0 0 0 8 0 0 0 4 6 0 0 7 3 0 0 0 6 0 4 0 0 0 0 9 0 0 5 0 6 0 0 6 0 0 0 9 0 0 5 9 0 0 0 5 0 0 7 2 0 0 8 6 7 0 0 1 0 0 7 6 0 0 0 3 0 0	678540 nanoseconds = 0.6 milliseconds	82 operations

Medium Inputs

7 0 0 8 0 0 9 0 3 0 0 0 2 0 0 0 1 8 0 0 1 0 0 3 0 0 0 0 1 5 0 0 0 3 4 0 2 0 0 0 0 8 6 0 1 0 0 3 0 7 4 0 0 0 0 0 6 3 0 0 0 8 0 0 3 0 0 0 5 1 0 9 9 5 0 0 8 0 0 0 0	834085 nanoseconds = 0.8 milliseconds	178 operations

4 0 0 8 0 0 0 0 0 0 5 0 0 2 7 0 0 1 2 0 0 0 0 0 3 0 7 0 0 2 0 1 0 5 0 0 0 0 0 9 0 2 0 0 4 1 2 0 0 0 0 0 6 0 0 7 0 2 8 0 1 0 0 0 6 1 5 0 0 0 4 0 0 2 0 0 0 1 6 3 0	7720505 nanoseconds = 7.7 milliseconds	13884 operations
5 0 7 0 0 9 0 0 0 0 4 0 0 0 5 0 9 2 0 0 2 0 0 0 3 5 0 0 0 0 0 9 0 8 4 1 0 1 4 0 8 0 0 0 0 2 8 0 7 1 0 0 0 0 0 0 0 9 6 0 0 8 0 0 2 5 0 0 0 0 0 9 3 0 0 0 0 8 7 0 0	1100161 nanoseconds = 1.1 milliseconds	503 operations

Hard Inputs

0 5 3 0 6 0 0 0 0 0 0 0 0 0 3 0 2 6 2 9 0 0 0 8 5 0 0 0 0 2 1 0 0 0 3 0 0 7 5 0 0 0 0 0 0 0 0 0 0 0 2 8 0 1 3 0 0 9 0 0 0 0 0 5 0 1 0 4 0 9 0 2 0 0 0 0 0 0 3 7 0	6623904 nanoseconds = 6.6 milliseconds	9741 operations
0 0 0 0 4 0 0 3 0 0 0 0 1 0 7 0 6 0 1 0 0 0 0 0 0 4 5 6 0 0 0 0 8 0 7 0 0 7 0 9 6 3 0 5 0 0 1 0 4 0 0 0 0 9 4 8 0 0 0 0 0 0 3 0 9 0 6 0 1 0 0 0 0 5 0 0 3 0 0 0 0	12554631 nanoseconds = 12.6 milliseconds	27694 operations

0 7 0 9 0 0 0 0 8 0 4 0 0 6 8 0 5 0 5 0 0 0 7 0 0 0 0 0 0 0 2 4 0 0 3 0 3 0 4 0 0 6 0 0 0 6 0 0 0 0 0 9 0 1 0 8 0 6 0 0 0 0 9 0 0 9 8 0 0 0 7 0 0 0 5 0 0 4 8 0 0	5966563 nanoseconds = 6.0 milliseconds	9127 operations

Very Hard Inputs

0 0 6 0 5 0 0 7 3 0 3 0 2 0 0 0 0 8 0 0 0 1 3 0 0 0 0 0 0 9 0 0 0 2 0 0 0 2 0 0 1 0 0 3 0 0 0 7 0 0 0 9 0 0 0 0 0 0 6 8 0 0 0 4 0 0 0 0 1 0 5 0 5 1 0 0 4 0 3 0 0	13851059 nanoseconds = 13.9 milliseconds	27040 operations
0 8 4 0 0 0 0 0 0 0 0 9 0 3 0 8 0 0 7 0 0 0 0 2 1 0 0 0 7 0 0 0 0 0 0 1 0 0 8 4 0 1 0 0 0 0 0 0 3 5 0 0 6 0 0 5 0 6 0 0 0 0 0 1 0 0 0 0 0 3 0 8 4 0 7 0 0 0 0 9 0	7422293 nanoseconds = 7.4 milliseconds	11355 operations
0 0 0 0 6 0 0 0 0 0 0 0 0 0 1 0 5 3 0 4 9 0 0 5 0 0 0 0 0 0 0 0 0 5 3 0 6 0 3 0 0 0 0 0 2 2 0 0 8 0 4 0 0 0 0 0 0 2 4 8 0 0 1 0 7 0 9 0 0 3 0 0 0 0 8 0 0 0 0 0 9	60450250 nanoseconds = 60.5 milliseconds	125574 operations

Solved Puzzle

2 9 6 8 5 4 1 7 3 7 3 1 2 9 6 5 4 8 8 4 5 1 3 7 6 9 2 1 8 9 4 7 3 2 6 5 6 2 4 9 1 5 8 3 7 3 5 7 6 8 2 9 1 4 9 7 3 5 6 8 4 2 1 4 6 8 3 2 1 7 5 9 5 1 2 7 4 9 3 8 6	169714 nanoseconds = 0.2 milliseconds	82 operations

Free Table Puzzle

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1087386 nanoseconds = 1.1 milliseconds	392 operations

One missing value Puzzle

1 2 3 4 5 6 7 8 0 4 5 6 7 8 9 1 2 3 7 8 9 1 2 3 4 5 6 2 1 4 3 6 5 8 9 7 3 6 5 8 9 7 2 1 4 8 9 7 2 1 4 3 6 5 5 3 1 6 4 2 9 7 8 6 4 2 9 7 8 5 3 1 9 7 8 5 3 1 6 4 2	187842 nanoseconds = 0.2 milliseconds	82 operations

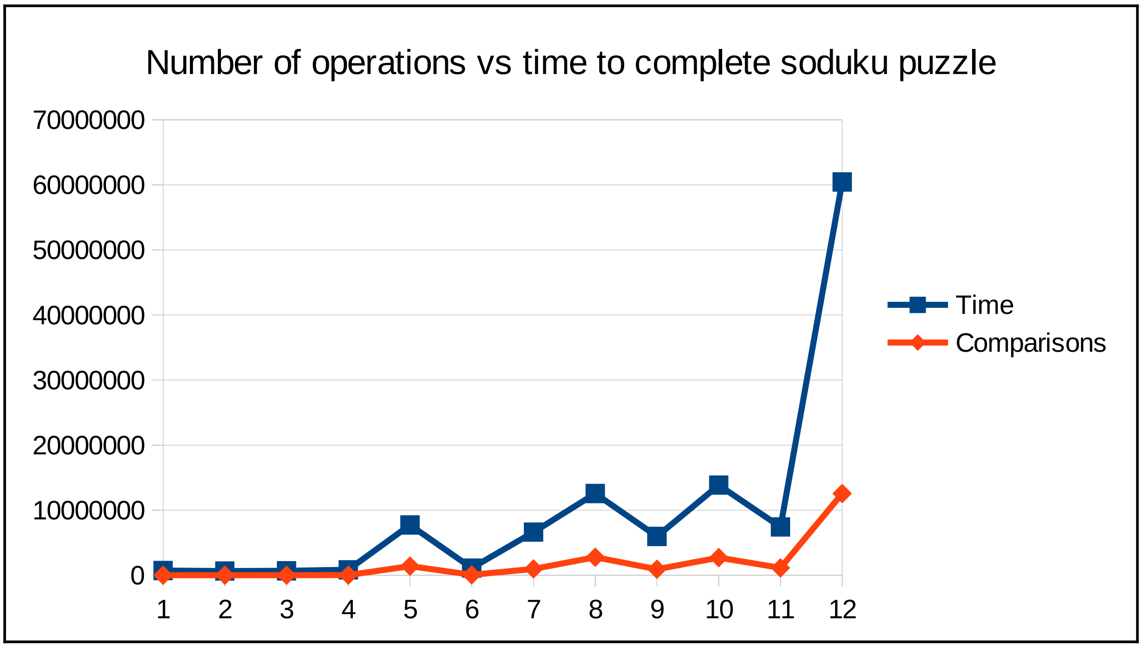
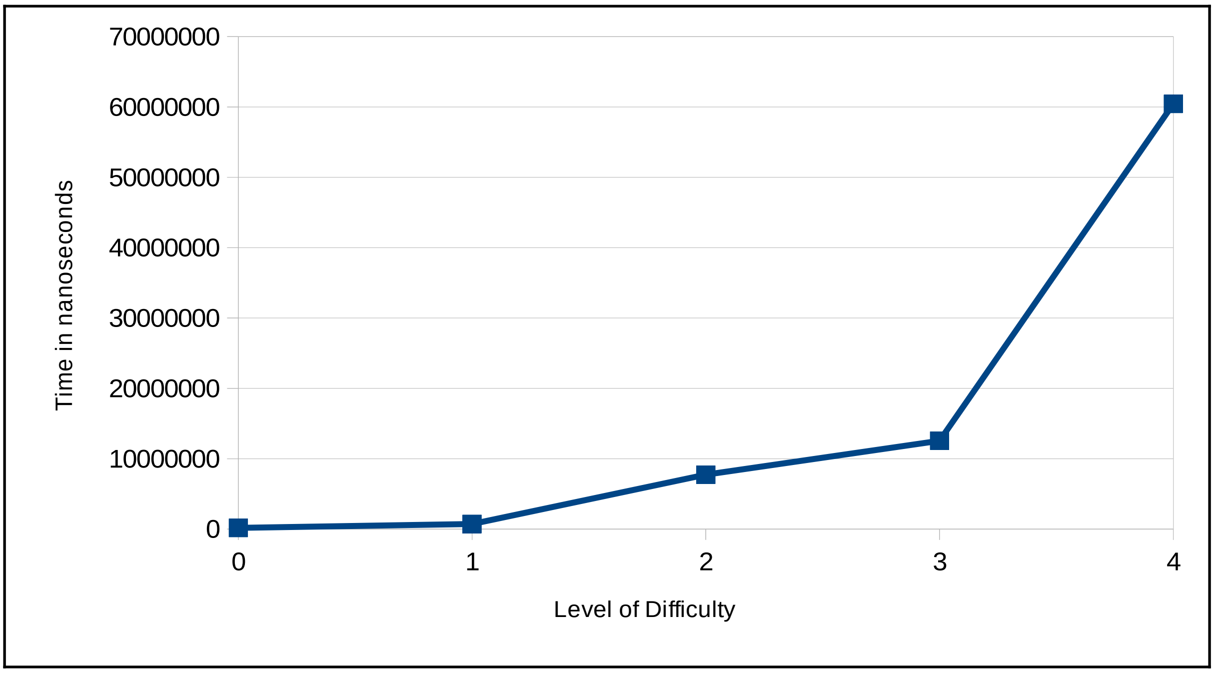


Table and Graph plot of highest time and operations from each level

Level of Sudoku puzzle	Time in nanoseconds	Number of operations
Solved Soduku = Level 0	169714	82
Easy = Level 1	712796	82
Medium = Level 2	7720505	13884
Hard = Level 3	12554631	27694
Very Hard = Level 4	60450250	125574
Single missing value Soduku	187842	82
Empty Soduku	1087386	392



Interpretation of results:

1) The higher the level of difficulty of the sudoku puzzle the longer the algorithm will take to solve the puzzle.

2) There is a strong positive correlation between the time taken to solve the puzzle and the number of operations done.

3) Time taken to solve the puzzle does not depend on the number of empty spaces or number of given clues but rather the level of difficulty of the puzzle.

Relate results to theory:

The theory that the time complexity is O(9^(n*n)), in this case being 1.97*10^77, is confirmed by the results as no number of operations to solve single sudoku puzzle reached 1.97*10^77, thus the equivalent time of 1.97*10^77 operations will not be reached due to the strong positive correlation between operations and time to solve puzzle.

Conclusion:

Solving Sudoku using a backtracking algorithm method, our algorithm would have to try each available number across all empty cells. Such an algorithm would have a runtime complexity of *O(9^(n²))*, where *n* is size of the Sudoku puzzle. The algorithm would perform *1.97*10^77* operations to find a solution. That is not practical. In practice the runtime would vary according to the difficulty of the puzzle itself and the number of options for each empty cell. Therefore the time complexity of O(9^(n*n)) is valid for the backtracking algorithm as it is the upper bound.

References:

https://www.101computing.net/backtracking-algorithm-sudoku-solver/

https://en.wikipedia.org/wiki/Sudoku

www.websudoku.com

www.sudokukingdom.com

https://medium.com/optima-blog/solving-sudoku-fast-702912c13307