Optimised Sudoku Solver.

Algorithm(s) and Approach(es).

I considered two approaches to the problem of solving partially complete 9x9 Sudoku grids: **(1)** fully optimised backtracking depth-first search and **(2)** Knuth's Algorithm X.

Overview of First Approach

Backtracking Depth-First Search – used recursive solution in which we check all assignments to a current variable (Sudoku cell), and for each inconsistent solution *(doesn't meet constraints of no duplicate cells in row, column or sub-square)* we backtrack and try new assignments (depth-first behaviour is due to use of recursive calls and the inherent call stack). Valid solutions are those where there are no more assignments to make (complete – no more zeroes on the grid) and every assignment is validated as consistent. ***(Full explanation given in notebook).***

Constraint Propagation – only continue search on consistent assignments, reduces search space by eliminating dead ends.

Minimum Remaining Values with Only-Choice – heuristic used to reduce search space at the variable selection level. Selects the next variable with the least number of assignments to check. If finds a variable leaving a single assignment, then immediately returns this as the *only-choice* case (since cannot have < 1 assignments).

Forward Pruning - reduces domain which LCV must order by eliminating the values in the domain which are already present in the current row, column and sub-square of the assigned variable.

Least Constraining Value – heuristic used to prune the domain of values to assign. Selects first the value which least constrains other variables (as measured by the number of possible assignments to the variables in the same row, column and sub-square as the value assigned).

Other optimisations

1. Leveraged NumPy vectorised operations to process the Sudoku matrix as opposed to direct iteration – more efficient due to use of GPU for parallelisation. Reduces linear complexity for array processing to a constant factor.
2. Create assignment once on first call by passing empty assignment as a parameter to the solver.

Complexity analysis / performance analysis

There is no theoretical change in time complexity from the standard backtracking brute force approach, compared to the use of optimisation techniques. However, my algorithm is measurably more performant than the standard approach.

1. Very Easy Puzzles – average speed of 0.01s w/o optimisations, average of 0.0001s with.
2. Easy Puzzles – average speed of 0.01s w/o optimisation, average of 0.0005s with.
3. Medium Puzzles – average speed of 0.1s w/o optimisation, average of 0.004s with.
4. Hard Puzzles – never solved w/o optimisations (past time limit), averaging 15s with.