Sudoku One Pager

Sudoku puzzles require solvers to assign values from the set $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ to each of the $n^2 \times n^2$ cells, where some values are provided, subject to constraints; the values in a cells row, column and box, $n \times n$ units, must all be different. This Sudoku solver began life as a simple implementation of backtracking search with constraint propagation, and became performant with the introduction of three search space pruning optimisations.

The algorithm was developed in two phases, a foundation and an optimisation phase. The foundation phase involved building the search and constraint propagation mechanism. The algorithm searches depth-first, building a virtual tree by recursively visiting unassigned variables and trying one or more domain values, backtracking to an earlier unvisited node if a solution cannot be found. Each time a variable assumes a domain value, the other variables in the same row, column and box can no longer assume that value. This is the basis of constraint propagation and the algorithm ensures both node and arc consistency (Russell and Norvig, 2022 p.170).

The optimisation phase involved researching suitable heuristics to prune the search space, as well as optimising the code to reduce per node complexity. The minimum remaining value (MRV) and least constraining value (LCV) (Russell and Norvig, 2022 p.177) heuristics were implemented along with the Maintaining Arc Consistency (MAC) (Russell and Norvig, 2022 p.178) algorithm, and their impact on reducing the problem complexity is measured in Table 1.

The MRV heuristic had the most significant effect on average time to solution, reducing it from effective infeasibility to $\sim 500ms$. The LCV heuristic reduced solving time by up to $\sim 100ms$ and was relatively ineffective. MAC lead to a saving of $\sim 200ms$, but could not be tested independently due to implementation idiosyncrasies. MRV enables the algorithm to explore the most contrained variables first, calculating the least promising search paths as early as possible. For a given variable, LCV chooses the domain values imposing the least constraints on the other variables in its constraint space. This means that as the search progresses, unassigned variables are more likely to have valid domain values remaining. MAC prunes the search space by recursively enforcing arc consistency on highly constrained variables.

Heuristic	**Mean (ms)**	**Max (ms)**
None	$> 30,000$	$> 30,000$
LCV	$> 30,000$	$> 30,000$
MRV	503	1,238
MRV and LCV	460	1,199
MRV and MAC	297	685
MRV, LCV and MAC	198	649

Table 1: Shows the mean and max time to solution for the provided 15 "Hard" class problems for the base algorithm with various combinations of heuristics applied.

The worst case time complexity of this algorithm is $O(kd^n + p)$ and the worst case space complexity is $O(dn^2)$, where d is the number of domain values, n the number of variables and $k = n + 3nd + 4d + d\log(d)$, where n is associated with MRV complexity, $2d + d\log(d)$ with LCV complexity, nd with constraint copying, $2nd$ with constraint propagation and p with other one-time initialisation costs. Asymptotic worst case time and space complexity is $O(d^n)$ and $O(dn^2)$ respectively, although search space pruning means the algorithm does not take exponential time in practice.

The data in Table 1 show that the algorithm has been a success; it solves all tested puzzles and is reasonably performant with an average and worst time to solution under $200ms$ and $650ms$ respectively. However, Sudoku puzzles can be solved in significantly less time (Edinburgh, 2024). Therefore, there are three key improvements worth investigating. Firstly, implementing Sudoku specific heuristics such as the "Naked Triple" (Zambon, 2011), or intelligent backtracking methods such as conflict-directed jumping (Russell and Norvig, 2022 p.180), would further prune the search space and reduce the average time to solution. Secondly, per node complexity could be further optimised by, for example, using more efficient data structures and operations, such as representing the puzzle as a binary integer and using bitwise operations. Finally, switching to a different optimisation paradigm such as mixed integer programming might unlock a level of performance not possible with the current approach (Edinburgh, 2024).

References

Zambon, G., 2011. *Sudoku explained*. Lulu Com.

Russell, S. and Norvig, P., 2022. *Artificial intelligence: a modern approach*. Pearson Education Limited.

Edinburgh, T.U. of, 2024. *Sudoku modelling and solution* [Online]. Available from: `https://www.maths.ed.ac.uk/hall/Sudoku.pdf` [Accessed January 25, 2025].