Research Computing Coursework Report

James Hughes

December 11, 2023

Introduction

Explain some basics of Sudoku (citation!), and why this problem relates to/necessitates software development. Some salient points such as uniqueness of solution, 17 clues minimum [1] and so on. Basic definitions as in the documentation.

Algorithm Selection and Prototyping

I decided to base my solution on the "templates" method combined with "backtracking" as described on the Wikipedia page about Sudoku solving algorithms [2]. This is a brute-force method, which relies on the fact that the rules of Sudoku inherently limit the number of possible arrangements of any given digit into nine cells on the grid. We can see this using a combinatorial argument filling in the 3x3 boxes from left-to-right then top-to-bottom: in the first box there are 9 choices, then 6 choices (one row already occupied), then 4 choices (two rows), then 6 choices (one column) and so on.

$$\text{Number of possible templates} = 9 \cdot 6 \cdot 3 \cdot 6 \cdot 4 \cdot 2 \cdot 4 \cdot 2 \cdot 1 = 46656$$

The method involves generating all such templates, and then finding the subset of templates that are valid in the context of each digit, with respect to the given clues. Then, based on the remaining possibilities, we search through all possible combinations of entries of the empty cells. I opted for a brute-force algorithm since this would be robust; since all possibilities are considered, a solution will be always found so long as it exists. Moreover, due to the simplicity of the Sudoku-solving problem, even in the worst case (as I discuss later) brute-force does not take too long provided the code is optimised. In particular, this method has a far smaller search space than backtracking without using templates, and so should be faster, so long as finding the subsets of templates is implemented to be fast.

In my original plan for the software, the first consideration was the memory required to store all the template arrays, which is the most memory demanding data structure in my solution. Assuming the default NumPy behaviour of storing integer arrays as 64-bit signed integers, this would require $46656 \cdot 81 \cdot 8 \approx 3 \cdot 10^7$ bytes, or 30MB, which poses no issues. I decided to store the abstract representation of the grid as a `np.ndarray` type of shape

`(9, 9)` since this greatly improves the speed of the high number of mathematical operations performed on the grid to find the solution. However I also anticipated that this could have been subject to change as I started to develop the code. For this reason, I decided to code the solving part of the code first, and then the next major parts, namely handling the input and output of data, could be coded according to the final choice of representing the grid internally. It appeared that the two main parts of the code were the solving and then the input and output of data, so I decided that a reasonable modular approach would be to create two separate modules, `solve.py` and `data.py` under the package `sudokutools`. Finally, I also noted that significant error catching was required somewhere between the input and solve stages, primarily because beyond 'obvious' violations from inputs that do not resemble a sudoku grid, there is the more subtle violation of a grid that does not have a solution, and could therefore cause the solver code to continue indefinitely if not caught.

INSERT DIAGRAM

Development, Experimentation and Profiling

Mention of testing-led Development

Git: Branches main, dev, test. Commit messages. Gitignore as

Started with templates

Refactoring

Commenting

First solve algorithm, brute-force of combinations

Extra template filtering

Proper backtracking

Read and write

Error catching

Profiling using line profiler in dev

Optimisation in test branch

Validation, Unit Tests and Continuous Integration

Validation ?

Testing: reason for multiple files
pre-commit: testing excluded, changes to the args

Packaging and Usability

Structure of packages, modularity, refactored code routines
Usability: docker, documentation, error catching?

Summary

Reasons for software development practices
Lessons learnt?

References

[1] wikipedia.org. Mathematics of sudoku, 2023. Date accessed: 7/12/2023.

[2] wikipedia.org. Sudoku solving algorithms, 2023. Date accessed: 7/12/2023.