Assignment 4 - Solving Constraint Satisfaction Problems

Morgan Heggland & Patrik Kjærran - MTIØT
TDT4136 - Introduction to Artificial Intelligence

October 9, 2019

1 Introduction

Assignment 4 in TDT4136 revolves around algorithms for solving constraint satisfaction problems, applied to Sudoku. The assignment asks us to implement a generic CSP solver and use it to solve Sudoku boards. In the following sections we will describe our design and approach to the exercise as well as the results we obtained.

2 Deliverables

In addition to the report, a .zip file containing the source code for our CSP solver follows. This includes `assignment_base.py` and `assignment_impl.py`. For ease of access, the boards are also included.

3 CSP solver implementation

The solver is implemented based on the backtracking algorithm described in the course syllabus [1]. The algorithm recursively assign values to variables in an attempt to satisfy the given constraints. To reduce the domain for a given variable assignment, inference based on the AC-3 algorithm is used.

The code is split into two files, `assignment_base.py` and `assignment_impl.py`. The first file contains an abstract base class, `CSP`, capable of encapsulating any constraint satisfaction problem described in terms of variables, domains and constraints. `assignment_impl.py` contains the class `CSPImpl`, which extends the base class, implementing the following methods:

- BACKTRACK: Performs a single iteration of variable assignment and recursively solves the subproblem.
- ORDER-LEGAL-VALUES: Decides the order in which a variable's legal values should be assigned according to a heuristic.

- SELECT-UNASSIGNED-VARIABLE: Selects an unassigned variable according to a heuristic.
- INFERENCE: Given a set of assignments and constraints, revises the corresponding variable domains as well as the domain of any variables affected by the domain reduction.
- REVISE: Given a pair of variables and their domains, removes all values which violates a constraint in the former variables' domain.

We have implemented ORDER-LEGAL-VALUES using the *least constraining value* heuristic, meaning the values that rule out the smallest number of legal values in neighbouring variables are ordered first. This ensures that values assigned at a given moment leaves more "flexibility" for later assignments. This heuristic resulted in only a slight increase in performance, reducing the number of inferences needed. It did however increase the number of calls to CALLBACK in one of the boards, compared to returning legal values in increasing order.

SELECT-UNASSIGNED-VARIABLE is implemented using the "degree" heuristic, meaning the variable with the highest number of constraints is selected. For a broad range of constraint satisfaction problems, this tends to lead to quicker detection of dead end branches.

4 Results

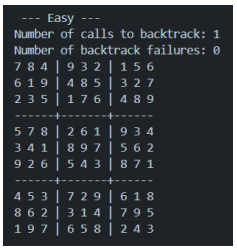
The implementation described above resulted in a very capable CSP solver, solving the Sudoku boards with ease. In order to measure performance, the algorithm counts the number of recursive calls to `backtrack` as well as number of dead ends reached (failures). These values are printed alongside the solutions for each board. The results can be seen in Figure 1 below.

We see that for the easy board, constraint-checking by the AC-3 algorithm is sufficient for finding an unique solution, explaining why the solution was found within the very first iteration.

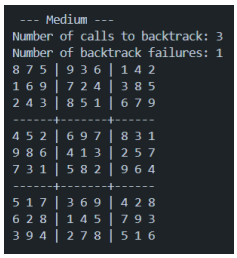
5 Appendix

After creating our Sudoku solver, we were curious to test its capabilities. Seeing as it solved the hardest provided board with ease, we wanted to test the solver on even harder boards. For a computer, the main difficulty lies in the number of preassigned values, dictates the branching factor. It has been mathematically proven that a board with 16 or less starting values is not solvable [2]. Therefore, we tested a sample board found online with 17 values. This was no match for our solver, which found the solution in 2.1 seconds. The starting board and results can be seen in Figure 2.

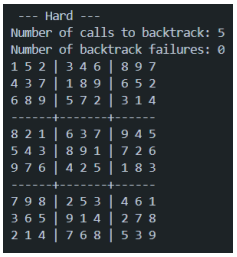
Another board found online [3] claimed to be the "world's hardest board", so we tried that as well. This board took approximately 5 seconds to solve, but was still no match for our solver, proving its efficiency. For this puzzle, the



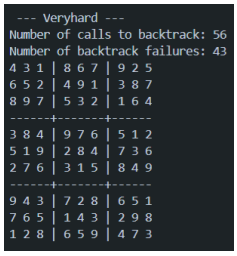
(a) Solution to the easy board



(b) Solution to the medium board



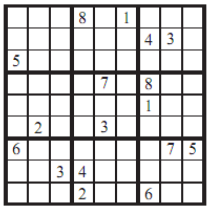
(c) Solution to the hard board



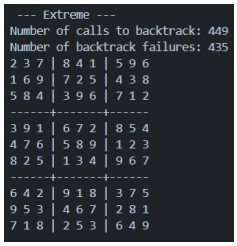
(d) Solution to the very hard board

Figure 1: The solutions to the provided problems.

computer is required to look several moves ahead even after perfectly deducing the legal values for each cell, which results in encountering a lot of dead ends. The results can be seen in Figure 3.



(a) The sample 17-digit board unsolved



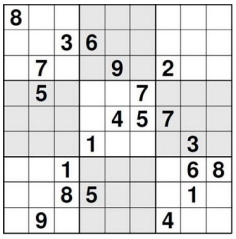
(b) Solution to the 17-digit board

Figure 2: Solving a 17-digit board.

References

[1] S. J. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2010.

[2] G. McGuire, B. Tugemann, and G. Civario, "There is no 16-clue sudoku: Solving the sudoku minimum number of clues problem," *CoRR*, vol. abs/1201.0749, 2012. [Online]. Available: http://arxiv.org/abs/1201.0749



(a) The "world's hardest" board unsolved

```
 --- Worldshardest ---
Number of calls to backtrack: 919
Number of backtrack failures: 905
8 1 2 | 7 5 3 | 6 4 9
9 4 3 | 6 8 2 | 1 7 5
6 7 5 | 4 9 1 | 2 8 3
------+-------+------
1 5 4 | 2 3 7 | 8 9 6
3 6 9 | 8 4 5 | 7 2 1
2 8 7 | 1 6 9 | 5 3 4
------+-------+------
5 2 1 | 9 7 4 | 3 6 8
4 3 8 | 5 2 6 | 9 1 7
7 9 6 | 3 1 8 | 4 5 2
```

(b) Solution to the "world's hardest" board

Figure 3: Solving the "world's hardest" board.

[3] N. Collins, "World's hardest sudoku: can you crack it?" 2012. [Online]. Available: https://www.telegraph.co.uk/news/science/science-news/9359579/Worlds-hardest-sudoku-can-you-crack-it.html