Ranking Sudoku Difficulty

Uzi Friedman 204030746

July 2020

1 Introduction

I've chosen to try and give a difficulty ranking for different Sudoku puzzles as my final project. When given a legal Sudoku board, the function outputs an integer score representing the estimated difficulty of the puzzle - the higher the score the harder the puzzle. The score is calculated based on the number of different uncertain decisions one needs to make during the solving process. I'm using my own (admittedly limited) Sudoku solving techniques as the solving heuristic used in the algorithm.

2 Scoring Algorithm description

When given a legal Sudoku puzzle, the algorithm iterative tries to find all cells who's values can be deterministic decided based on all other cell's values. To do so, it iterates over all cells and uses SAT4J to find all possible values for that cell which satisfy the conditions for a legal puzzle. If a cell has only a single value possible, that it is assigned that value. Once no more such cells exist, the algorithm attempts to deduce the value of cells based on other already assigned cells. For every row, column and box in the puzzle, it again generates a list of possible values for all cells in that object. If any cell has a possible value that is unique to it (no other cell in the object can be assigned that value) then that value is assigned to it. After deducing all possible cells, it loops back to finding cells with a single possible value. We assume that this kind of process is easy an does not contribute to the difficulty score of a puzzle. Once both procedures can't find any more cells to assign values to, the algorithm is forced to guess the value of a cell. It selects one of the cells with the fewest possible values an increases the difficulty score by the number of possible values times the recursion depth of the current call to the function. It then adds to the difficulty score the scores returned by recursively calls itself for each possible value, each time new puzzles created by all cells assigned values so far, and with the selected cell assigned with the currently tested value. The function returns if a solution is found, or if a contradiction is detected in the current puzzle (by guessing the wrong value to be assigned to a cell).

3 Creating Easy Puzzles

In order to search for an easy (difficulty score of 0) puzzle with as little hints as possible, we start by creating a complete legal Sudoku board. We do this by using SAT4J in order to solve a Sudoku problem without any 'hint' constraints (obviously there are many solutions). Once we have such a board, we start removing elements at random. After each removal, we check whether or not the board is still a legal Sudoku puzzle (again using SAT4J and checking if there exists more than 1 solution. We know that there is at least one - the initial one). If it is still a legal puzzle, and the difficulty score is still 0, we keep the removal, otherwise we set the cell's value to what it previously was. The number of times we choose a different random cell to try and remove is equal to the total number of cells in the board. Increasing this number might find solutions with fewer hints, as different selections can produce better boards, but we limit ourselves due to run-time complexity concerns. Once we are done removing cell values, we score the produced (legal) board using the function described above.

We compare the number of hints used in it to the number of hints used in the best puzzle found so far, and if it is fewer than it is the new best. Because of the way we generate the initial board is deterministic, it might be that other boards exists which can be better minimized and produce puzzles with fewer hints and a difficulty of 0. To account for that, we run the algorithm multiple times, each time requiring that the board used will be different than the last one used. Once the process is finished, we return the Sudoku puzzle board which has the lowest difficulty found and the minimum number of hints which produces it. Using this method, producing very difficult boards requires a very simple tweak to the algorithm. We remove the requirement to keep the difficulty at 0 when removing a value from a cell and When comparing boards to best found so far, optimize for maximising difficulty in addition to minimizing hints.