1 Introduction 1.1 What is Sudoku?

A Sudoku puzzle is a n-by-n grid that contains numbers from 1 to n, with box size n × n.

A standard Sudoku contains 81 cells, in a 9 × 9 grid, and has 9 boxes (3 × 3 grid) The goal of the puzzle game is to fill in the empty cells on the board such that each column, row,

and box (or called “subgrid”, “region”, “block”) contains every number in the set {1, ..., 9} exactly

once. Sudoku puzzles usually come with a partially filled-in board (clues). The difficulty of the

puzzle varies depending on how many numbers are given, as well as the location of the given

numbers. Solutions might not be unique, or might not even exist. However, a properly formulated Sudoku

puzzle should have an unique solution that can be reached logically.

1.2 Rules of Sudoku

• Each number must appear exactly once in each row.

• Each number must appear exactly once in each column.

• Each number must appear exactly once in each box.

The above rules imply no duplicate numbers in any row, column and box.

2 Solving Algorithms 2.1 Naive brute-force algorithm

A brute-force algorithm would enumerate all possible combinations of numbers for each empty

cell. This algorithm has an incredibly large search space to wade through. For instance, a blank

n-by-n grid has a total of nn×n different possible combinations of solutions. Solving Sudoku

puzzles in this way is extremely slow and computationally intensive, but guarantees to find a

solution eventually and the solving time is mostly unrelated to degree of difficulty.

2.2 Backtracking algorithm

A common algorithm to solve Sudoku boards is called backtracking, which is a type of brute-force

search. This algorithm is essentially a depth-first search (DFS) by completely exploring one

branch to a possible solution before moving to another branch.

Given a partially filled-in (incomplete) board, the following illustrates the backtracking algorithm

\1. Finds the first empty cell on the given board.

\2. Attempts to place the smallest possible number (namely 1) in that cell.

\3. Checks if the inserted number is valid. • If the number is valid proceed to the next empty cell and recursively repeat steps 1-3.

• If the number is not valid, increment its number by 1 and repeat step 3. If a cell is discovered where none of the possible numbers is allowed, then reset the cell you just filled to zero/blank and backtrack to the most recently filled cell. The value in that cell is then incremented by 1 and repeat step 3. If the number still cannot be incremented, then we backtrack again.

\4. Repeats the procedure until either there are no more empty cells on the board, which means we have found the solution. Or we backtracked to the first unfilled cell. In this case, no solution exists for the given board.

Rather than trying to continue a solution that can never possibly work which we do with naive brute-force algorithm, we only continue solutions that currently work. If they don’t work, we backtrack to the previous step and try other numbers again. This is going to be much faster than trying every single possible combination of solutions as naive brute-force algorithm does.