Simran Gidwani

CS1571

Fall 2019

9/4 Homework

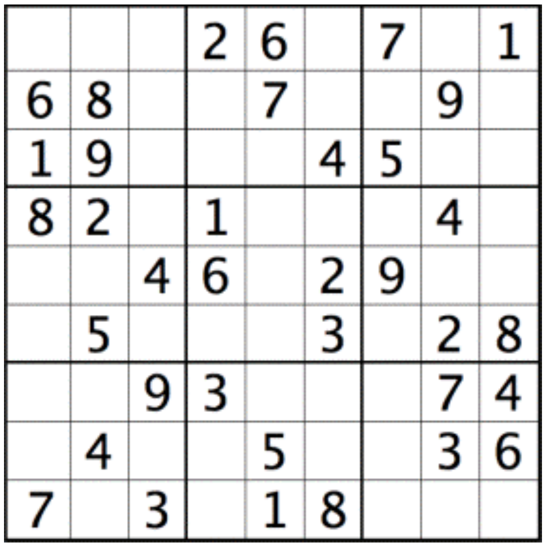
Introduction

Read Russell & Norvig, Chapters 3.1-3.4.

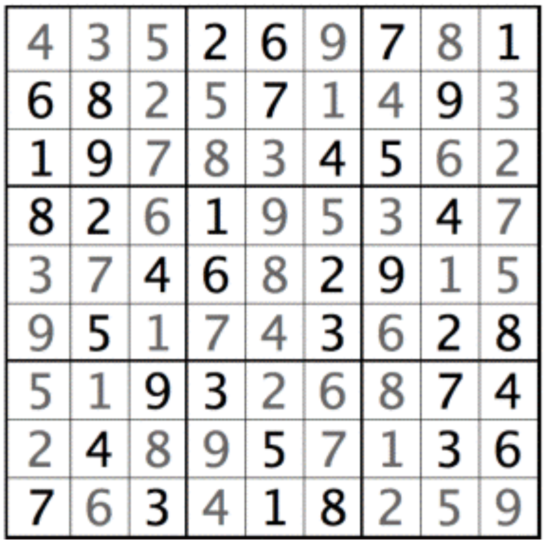
- 3.1-3.3 is mostly a review of what we covered in class last week. There is some information on complexity and the implementation of search that we didn't cover in depth; we will go over this in "Uninformed Search".
- 3.4 will be relatively new for now, but over the course of the next two weeks, we will be covering all the search algorithms discussed in 3.4.

You are going to explore algorithms for solving a Sudoku puzzle. The object of Sudoku is to put a digit from 1 to N in each cell of the grid so that every row, column, and bold region contains each digit once, where N is the number of rows and columns in the puzzle. (Usually N = 9.)

Here is an example of a Sudoku puzzle and its solution.



			2	6		7		1
6	8			7			9	
1	9				4	5		
8	2		1				4	
		4	6		2	9		
	5				3		2	8
		9	3				7	4
	4			5			3	6
7		3		1	8			



4	3	5	2	6	9	7	8	1
6	8	2	5	7	1	4	9	3
1	9	7	8	3	4	5	6	2
8	2	6	1	9	5	3	4	7
3	7	4	6	8	2	9	1	5
9	5	1	7	4	3	6	2	8
5	1	9	3	2	6	8	7	4
2	4	8	9	5	7	1	3	6
7	6	3	4	1	8	2	5	9

A. Problem Formulation

Last class we talked about formulating a problem as a search problem, and Sections 3.1 - 3.3 of the textbook discuss this as well. Represent the Sudoku problem as a search problem by defining the initial state, possible actions, transition model, goal test, and path cost. *Hint: You can follow the same approach used in the 8-queens problem to represent a Sudoku.*

The initial state: the state at which the sudoku puzzle begins partially filled in.
Possible actions: Starting from the top leftmost empty box, trying numbers 1-9 in the empty spaces and placing the first number that matches the restrictions of the puzzle being that that same number cannot exist in the same column or row.
Transition Model: Returns the board with that specific number added to it
Goal Test: Each block in the sudoku puzzle is filled in and meets the restrictions of the completion of the board.
Path Cost: Each step costs 1, so the path cost is the number of moves you have to make until the goal state is reached.

B. Complexity

What is *b, d*, and *m* of your search tree? What is an upper bound on the size of your search space? Explain your reasoning.

Branch factor, **B,** would be 9
Depth, **D,** would be 9
Max length of any state searched, **M,** would be 81

C. Breadth-First Search or Depth-First Search
Is breadth-first search or depth-first search better to use for solving a Sudoku? Why? A good answer will demonstrate understanding of the relative performance of breadth-first and depth-first search for this particular problem.

I think for this particular problem Depth First Search would be the better algorithm to use to solve the Sudoku Puzzle. The idea behind Depth First Search is that it expands less paths. Due to the idea of backtracking, instead of expanding paths that we know do not reach our goal state, it will backtrack and try a different number in the previous node. This eliminates the number of paths created. On the other side, Breadth First Search looks at the root, expands to the next level and continues expanding every level before continuing to the next. However, we know that the run times of both algorithms are 2e where "e" is the number of edges in the graph, because DFS is creating less paths, I think using Depth First Search would be the better approach.