Homework Assignment 1: Search Algorithms

CS486/686 – Spring 2017

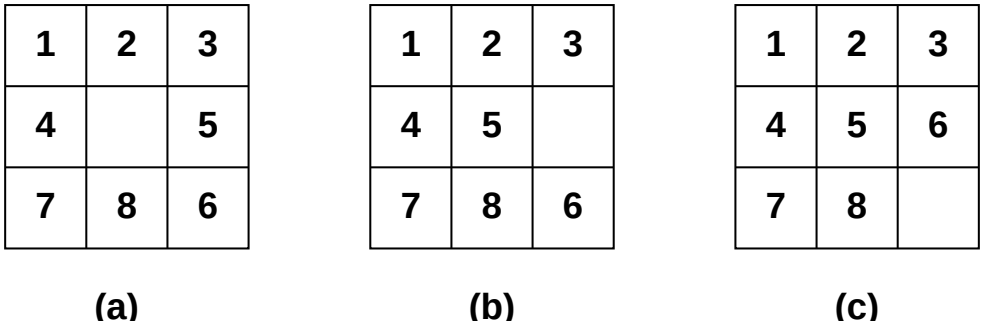
Out: May 8, 2017
Due: May 19 (11:59 pm), 2017.

Submit an electronic copy of your assignment via LEARN. Late submissions incur a 2% penalty for every rounded up hour past the deadline. For example, an assignment submitted 5 hours and 15 min late will receive a penalty of ceiling(5.25) * 2% = 12%.

Be sure to include your name and student number with your assignment.

1 Informed Search

Consider the 8-puzzle, which is a simple (one-person) game that we discussed briefly in class. In this game, tiles numbered 1 through 8 are moved on a 3-by-3 grid. Any tile adjacent to the blank position can be moved into the blank position. By moving tiles in sequence we attempt to reach the goal configuration. For example, in the figure below, we see three game configurations: the configuration (b) can be reached from configuration (a) by sliding tile 5 to the left; configuration (c) can be reached from configuration (b) by sliding tile 6 up. Configuration (c) is the *goal configuration*. The objective of the game is to reach the goal configuration from some starting configuration with as few moves as possible. Note that not all starting configurations can reach the goal.



1. **[20 pts]** Consider the following two heuristic functions that we discussed in class:

   - *Misplaced tile heuristic:* number of tiles (excluding the blank) that are misplaced (with respect to the goal configuration)
   - *Manhattan distance heuristic:* total Manhattan distance of all the tiles (excluding the blank). That is, for each tile, the Manhattan distance is the sum of the horizontal and vertical distances between its current position and the desired position in the goal configuration.

   Suppose that you use the above heuristics in A* search to solve the 8-puzzle game.

(a) Which heuristic do you expect to perform best? You do not need to implement the heuristics with A*. Based on the properties of each heuristic, just explain which one you expect to perform best.

(b) We have seen in class that those heuristics are *admissible*. Are they also *consistent*? Give a proof or a counter example for each heuristic.

2. **[20 pts]** IDA* combines iterative deepening with A* search. Answer the following questions regarding the expected performance of IDA* (you do not need to implement IDA*).

   (a) What is the time and space complexity of IDA*?

   (b) Is IDA* complete? Explain briefly.

   (c) Is IDA* optimal? Explain briefly.

2 Constraint Satisfaction

Using the language of your choice, you will implement a program that solves Sudoku puzzles using backtracking search. In a Sudoku puzzle, the goal is to fill a 9-by-9 grid so that each column, each row, and each of the nine 3-by-3 boxes (also called blocks or regions) contains the digits from 1 to 9, only one time each. In other words, a solved Sudoku puzzle has each digit from 1 to 9, exactly once in each row, once in each column and once in each of the nine 3-by-3 boxes. An example is shown below. The grid on the left is a starting configuration, and the grid on the right is the solution. You may not change any of the digits that are in the start configuration. A good source for information about Sudoku puzzles is http://www.websudoku.com.

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

(a) Starting configuration

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

(b) Solved puzzle

1. **[15 pts]** How did you formulate the Sudoku puzzle as a constraint satisfaction problem (CSP)?

   What to hand in: a detailed description of all variables, domains and constraints that are sufficient to model Sudoku as a CSP.

2. **[45 pts]** Implement a Sudoku solver (for 9-by-9 puzzles only) using the CSP formulation that you came up with. More precisely, implement the backtracking seach algorithm, forward checking and the three CSP heuristics discussed in class (e.g., most constrained variable, most constraining variable and least constraining value). For more details about these algorithms, review the lecture slides and read pages 141-145 (2nd edition) or pages 214-218 (3rd edition) of the textbook.

   You should implement the following combinations:

   - B: basic backtracking search (no forward checking, random variable order and random value order)
   - B+FC: backtracking search with forward checking (random variable order and random value order)
   - B+FC+H: backtracking search with forward checking and the 3 heuristics to order variables and values (break any remaining ties in the order of the variables and values at random)

Download the test puzzles (easy, medium, difficult and evil puzzles) posted on the course website. Note that the label (easy, medium, difficult and evil) is meant for humans and therefore may not reflect the level of difficulty for a computer program. Run each of the above combinations on each of the test puzzles 50 times and report the following information for each test puzzle:

- time to complete each puzzle (average and standard deviation of the 50 runs)
- number of nodes expanded (average and standard deviation of the 50 runs). A node is expanded when it is removed from the queue and its children are added to the queue (the children are not expanded until they are removed from the queue).

You should hand in two tables (one for time and the other one for the # of nodes). If your algorithm takes too long to run, reduce the number of runs from 50 to something feasible and simply indicate the number of runs used to compute each average and standard deviation (you will not be penalized if you report less than 50 runs).

Time

	B	B+FC	B+FC+H
Easy	avTime $\pm$ stdTime	avTime $\pm$ stdTime	avTime $\pm$ stdTime
Medium	avTime $\pm$ stdTime	avTime $\pm$ stdTime	avTime $\pm$ stdTime
Difficult	avTime $\pm$ stdTime	avTime $\pm$ stdTime	avTime $\pm$ stdTime
Evil	avTime $\pm$ stdTime	avTime $\pm$ stdTime	avTime $\pm$ stdTime

# of Nodes

	B	B+FC	B+FC+H
Easy	avNodes $\pm$ stdNodes	avNodes $\pm$ stdNodes	avNodes $\pm$ stdNodes
Medium	avNodes $\pm$ stdNodes	avNodes $\pm$ stdNodes	avNodes $\pm$ stdNodes
Difficult	avNodes $\pm$ stdNodes	avNodes $\pm$ stdNodes	avNodes $\pm$ stdNodes
Evil	avNodes $\pm$ stdNodes	avNodes $\pm$ stdNodes	avNodes $\pm$ stdNodes

What to hand in:

(a) Your code

(b) Solution for each test puzzle

(c) Tables showing your algorithm's performance in terms of time and number of nodes on the 4 test problems

(d) Brief dicussion to explain why your algorithm performed the way it did (max one page).