15-381 Artificial Intelligence, Fall 2008
Homework 1
Out: Sep 4, 2008
Due: Sep 18, 2008

Instructions

This assignment is due on Thursday, September 18, 2008. The written portion must be turned in at the beginning of class at noon on September 18th. Type or write legibly; illegible submissions will not receive credit. Write your name and Andrew ID clearly at the top of the assignment. The programming portion of this assignment must be electronically submitted by noon on September 18th. The submission instructions are described in the Programming section.

Collaboration Policy. You are to complete this assignment individually. However, you are encouraged to discuss the general algorithms and ideas in the class in order to help each other answer homework questions. You are also welcome to give each other examples that are not in the assignment in order to demonstrate how to solve problems. But we require you to:

- not explicitly tell each other answers;
- not copy answers;
- not allow your answers to be copied.

In those cases where you work with one or more other people on the general discussion of the assignment and surrounding topics, we ask that you specifically record on the assignment the names of the people you were in discussion with (or "none" if you did not talk to anyone else). This will help resolve the situation where a mistake in general discussion led to a replicated error among multiple solutions. This policy has been established in order to be fair to everyone in the class. We have a grading policy of watching for cheating and we will follow up if it is detected.

1 Written questions

1. Choose a news article after Aug 1. Briefly describe how AI has played a role in solving a problem mentioned in the article, or how you propose to use AI in such a problem. Your answer should be at least 500 words. (5 points)
2. This question will give you an excuse to play Sudoku (see www.websudoku.com for explanation) while doing homework. Consider using search to solve Sudoku puzzles: You are given a partially filled grid to start, and already know there is an answer.
   (a) Define a state representation for Sudoku answer search. A state is a partially filled, valid grid in which no rows, column, or 3x3 square contains duplicated digits. (3 points)
   (b) Write a pseudocode for the successor function (generate a list of successor states from the current state) given your representation. You can assume you have functions like "duplicated" that returns true if there are two identical non-empty elements (or non-zero digits) in a collection. (3 points)

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

Figure 1: A sampel Sudoku puzzle with 28 digits filled initially.

(c) Write a pseudocode for goal function that returns true if a state is a goal. You can assume the state is reached by the successor function above so is a valid state. (2 points)

(d) If the puzzle begins with 28 digits filled, what is L, the length of the shortest path to goal using your representation? (2 points)

(e) On a typical PC, which search algorithm would you choose, BFS, DFS, IDS, A*, or IDA*? Why? Hint: which is larger, L or the average branching factor B? (5 points)

3. Decide whether the following statements are true or false. If true, explain why. If false, give a contradicting example. Recall that B is the average branching factor and L is the length of the shortest path from start to goal. (8 points)

(a) Bi-directional BFS is always faster than BFS when $B \geq 2$ and $L \geq 4$.

(b) A* search always expands fewer nodes than DFS does.

(c) For any search space, there is always an admissible and consistent A* heuristic.

(d) IDA* does not need a priority queue as in A*, but can use the program stack in a recursive implementation as in DFS.

4. Please provide two different and non-trivial A* heuristics, one for each of the following search problems: (8 points)

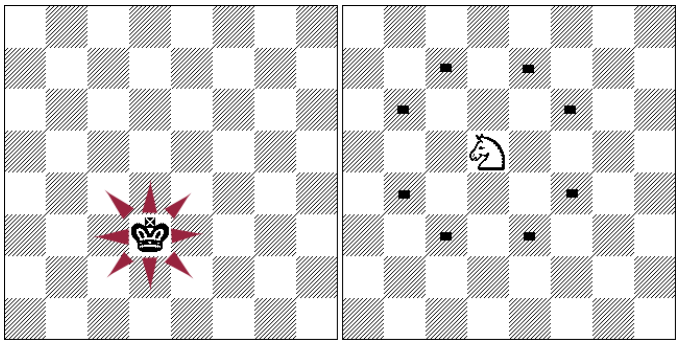

Figure 2: Moves of a king (left) and a knight (right) for question 4.

(a) In a 8x8 chess chessboard, move a king from top-left to bottom-right (no other piece in the chessboard). A king can move into one of the 8 adjacent squares. Propose an admissible and consistent A* heuristic; you can use x and y denote distance to bottom-right in x-axis and y-axis.

(b) In a 8x8 chess chessboard, move a knight from top-left to bottom-right (no other piece in the chessboard). See figures above for how a knight moves. Propose an admissible and consistent A* heuristic; you can use x and y denote distance to bottom-right in x-axis and y-axis.

5. Consider the problem of moving a knight on a 3x4 board, with start and goal states labeled as S and G in figure 3. The search space can be translated into the following graph. The letter in each node is its name and the subscript digit is the heuristic value. All transitions have cost 1.

S_3	H_1	D_1	K_1
I_1	J_2	A_2	E_2
C_1	B_2	G_0	F_3

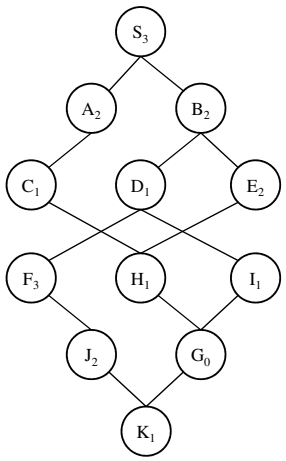

Figure 3: Search graph of knight on 3x4 board for question 5.

Make the following assumptions:

- All the algorithms do not generate paths with loops.
- Nodes are selected in alphabetical order when the algorithm finds a tie.
- A node is visited when that node is at the front of the search queue.

Write the sequence of nodes in the order visited by the specified methods. Note: You may find it useful to draw the search tree corresponding to the graph above.

(a) Depth-first search. (8 points)

(b) Breadth-first search. (8 points)

(c) A* search. In addition to the sequence of nodes visited, you are also to attach the estimated total cost f(s)=g(s)+h(s) for each node visited and return the final path found and its length. (8 points)

2 Programming

WARNING: This problem takes considerably more time than the rest of the assignment. Do NOT leave this until the last minute.

Escaping-Mellon-Maze: Your TA always get lost in the Mellon Institute and wish you to write a search program to escape the maze. You are given a 25x25 maze as shown in figure 4. The starting cell is

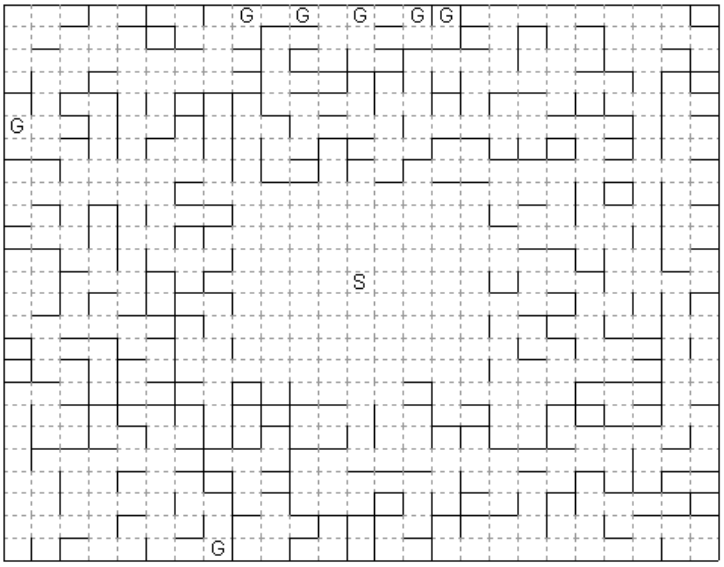

Figure 4: The 25 by 25 maze for the programming part. S is the starting cell and G are the goal cells. Included in the support archive as *mellon.maze*

.

labeled S and the goals are labeled G (reaching any goal cell is OK). You can only move in four directions: north, east, south, and west.

Matlab is strongly recommended, since helper functions are in Matlab. In the support file archive, we have provided the following helper functions for the maze and two maze files, the above maze and a small sample maze. Cells are indexed as in the 5x5 maze in figure 5, following Matlab implementation.

- *maze_load*: loads a maze into a structure for use with other function.
- *maze_draw*: provides a visualization of the maze.
- *maze_neighbors*: returns valid neighbors of a cell.
- *maze_start*: returns starting cell of a maze.
- *maze_isgoal*: returns whether a cell is one of the goal.
- *maze_XY_from_index*: gets the X and Y coordinates of the location at index.
- *maze_index_from_XY*: gets the index from the X and Y coordinates.
- *maze_drawpath*: draws a path on a figure created by *maze_draw*.

A priority queue library is also provided:

- *pq_init*: Initialize a priority queue.
- *pq_set*: Reset the priority of an element or insert it if it's not already there.

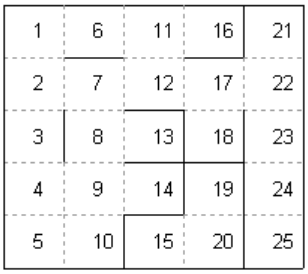

Figure 5: Cells are indexed from left to right and then top to down, as shown in this figure.

- *pq_pop*: Remove and return the first element (the first element is the one with the smallest numerical priority value). It breaks ties arbitrarily.
- *pq_test*: Check whether an item is already on the priority queue.

Type "help *functionname*" or see the source of helper functions for documentations. Call *maze_load* with the two included maze files, *mellon.maze* and *5x5.maze*, to load the mazes. Call *maze_draw(maze, 2)* to draw the maze, and call *maze_drawpath* to examine the path found by your implementation. Test your implementations on the 5x5 maze first. If you run into memory problem or infinite loop, setting a maximum number of expanded nodes to 10,000 might help debugging.

1. Implement and run BFS (breadth-first search). Consider neighbors in the following order when expanding nodes: north, east, south, and west. Report the run time, number of nodes expanded, path length, and the path found (as a list of indices) on the above 25x25 maze in the file *mellon.maze*. Use the interface provided in *bfs.m* and fill in your implementation. (10 points)
2. Implement and run DFS (depth-first search), based on the same rules and report the same items as above. Use the interface provided in *dfs.m* and fill in your implementation. (10 points)
3. Implement and run A* search, based on the same rules and report the same items as above. Propose and try *two* different admissible heuristics. Use the interface provided in *astar.m* and fill in your implementation. (10 points)
4. Compare the search results of the different methods and heuristics. Which methods and which heuristic performed best? Why? (10 points)

Do *not* attach your code to the writeup. Make a tarball of your code and output, name it <yourandrewid>.tgz, and copy it to
/afs/andrew/course/15/381-f08/submit/hw1/<yourandrewid>/<yourandrewid>.tgz