15-381 Artificial Intelligence, Fall 2008
Homework 1

Out: Sep 4, 2008
Due: Sep 18, 2008

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)

					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.

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)

   A 9x9 matrix whose elements are 1 to 9 or 0 as empty. Transitions are any *valid* filling of an empty cell. Only valid matrices (no duplication in rows, column, or 3x3 squares) are allowed.

   (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)

```
M = current state
for i = 1 : 9
    for j = 1 : 9
        if M(i,j) == 0
            copy M to A;
            for x = 1 : 9
                A(i,j) = x;
                if valid(A)
```

```
                    add A to successor-list;
                end
            end
        end
    end
end

function valid(A)
    for i = 1 : 9
        if duplicated(A(i,:)), return False; end
    end
    for i = 1 : 9
        if duplicated(A(:,i)), return False; end
    end
    for i = 1 : 9
        for j = 1 : 9
            if duplicated(A(i:i+2,j:j+2)), return False; end
        end
    end
    return True;
```

(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)

```
M = current state
if any(any(M==0)), return False; else return True; end
```

(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)

$$L = 81 - 28 = 53$$

(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)

DFS is the most suitable, and actually almost all Sudoku search program use DFS.

B can be in the range of 9×53 so $B >> L$. Hence BFS and A* will have serious memory problem on a typical PC.

L is fixed so $L = L_{\max} = L_{\min}$. BFS and A* makes no difference because both have to reach the final step ($L-1$ transitions) to find the goal. IDA* will be the same as IDS which only waste time in the first $L-1$ levels and did the same as DFS in the final L-th level. DFS is the only one among these algorithm that makes sense.

Many students did not get the correct answer for this question. Some suggested that A* is best with a good heuristics, while some others said they cannot come up with a good heuristics. In fact there is an exact A* heuristics, the number of empty cells, but it makes no difference because any transition will result in the same cost and heuristics. On the other hand there are tricks for DFS to find a solution faster (ex. fill in empty cells with fewer successors first) but it is not the kind of A* heuristics we talk about in class. Some chose IDA* or IDS, but there is no point when L is fixed.

The point here is that all search algorithms have unique advantages and you really need to understand your search problem to choose one; DFS is usually considered too simple and not powerful, but here is a problem in which DFS is the only reasonable choice.

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 $L \geq 4$.

   False. Consider searching on a tree from the leaf to the root, and transitions always move towards the root. BFS is fast because branching factor is always 1, while BIBFS waste time on the half of the reverse part tracing a lot of branches.

   Note that time complexity is no the same as actual timing or number of nodes expanded. $O(\cdots)$ are worst case, and under a constant factor. On the other hand in cases like A* being more efficient than any optimal search given a heuristic, we really mean ALWAYS — it has been proved that for any search space no other optimal search can expand fewer nodes. Know the difference. Also note that treating B as a constant is just an approximation, since actually the branching factor is different between nodes. Finally you need to understand the assumptions underlying the analysis that BIBFS is faster than BFS.

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

   False. No optimal search algorithm can be more efficient than A*, but DFS is not optimal and it can be lucky in searching goals. For example on the Sudoku search problem above, DFS almost always expand fewer nodes than A*, because A* need to expand $L - 1$ levels.

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

   True. For example $h(s) = 0$.

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

   True. Note that the inner loop in IDA* is DFS, not A*.

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

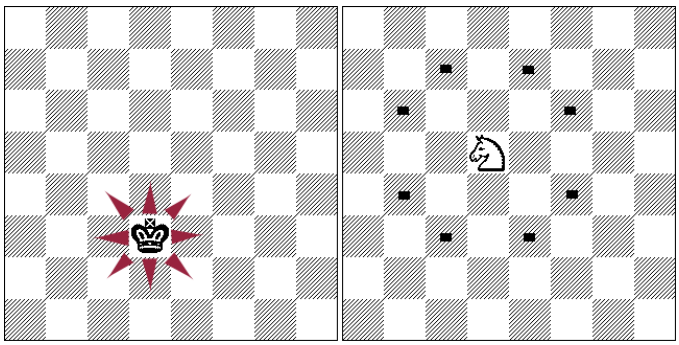


Figure 2: Moves of a king (left) and a knight (right) for question 4.

   (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.

   Possible heuristics include: $\max(x, y), (x + y)/2, \sqrt{x^2 + y^2}/\sqrt{2}$, etc.

   (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.

Possible heuristics include: $\max(x, y)/2, (x+y)/3, \sqrt{x^2+y^2}/\sqrt{5}$, etc. Some students wrote down the exact solution and got full score. However note that it missed the point of A* heuristic because it is the same as performing (reversed) BFS first and then we already know the solution.

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.

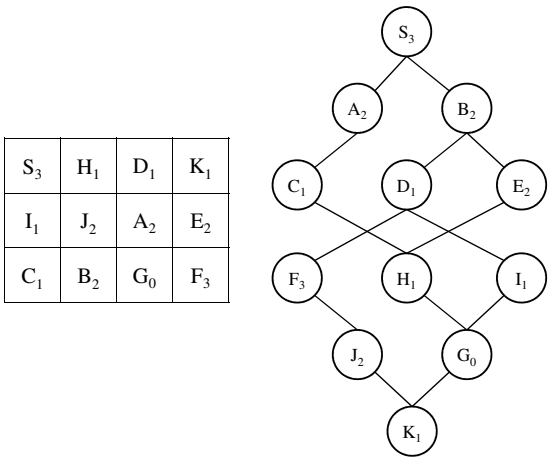


Figure 3: Search graph of knight on 3x4 board for question 5.

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)
S A C H E B D F J K G

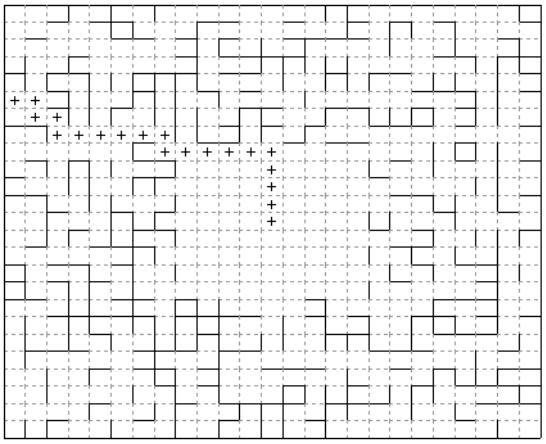
(b) Breadth-first search. (8 points)
S A B C D E H F I G

(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)
S (0+3=3), A (1+2=3), B (1+2=3), C (2+1=3), D (2+1=3), E (2+2=4), H (3+1=4), G (4+0=4)
Final path: SACHG

2 Programming

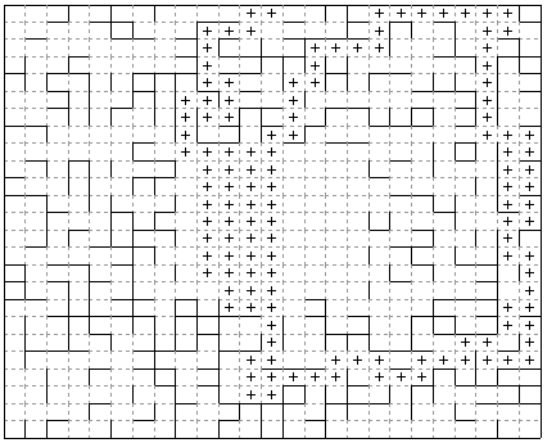
1. Implement and run BFS (breadth-first search). Consider neighbors in the following order when expanding nodes: north, east, south, and west. Report the run time, number of nodes expanded, path length, and the path found (as a list of indices) on the above 25x25 maze in the file *mellon.maze*. Use the interface provided in *bfs.m* and fill in your implementation. (10 points)

   BFS should find the optimal path as shown below. The length including the starting and goal cell is 20.



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)

   DFS should find the following winding path of length 129 (including starting and goal cells) if you following the required ordering of north, east, south, and west.



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)

The most common heuristics are minimum of Euclidean distances, $\min_g(|x - x_g| + |y - y_g|)$, and minimum of Manhattan distances, $\min_g \sqrt{(x - x_g)^2 + (y - y_g)^2}$.
You can also use minimum distance to all the walls, $\min(x - 1,\ y - 1,\ 25 - x,\ 25 - y)$, or even mix of the above, $\min(|x - 1| + |y - 6|,\ |x - 8| + |25 - y|,\ y - 1)$.

You should find the optimal path as the one in BFS, as long as the heuristic is truly admissible.

4. Compare the search results of the different methods and heuristics. Which methods and which heuristic performed best? Why? (10 points)

   The path found by DFS is far from optimal, though BFS expanded more nodes. A* finds the optimal path with the fewest nodes expanded, if the heuristic is good. For most students, DFS run fastest even if the number of expanded nodes are more than that of A*. This may be due to implementation details, Matlab being slow, and a small map size.