National University of Computer & Emerging Sciences

(Karachi Campus)

Midterm Examination I – Spring 2016-sol

Artificial Intelligence (CS401)

Time Allowed: 60 Min. Max. Points: 50

Dated: February 18, 2016

Instructions: Attempt all questions. Be to the point, there is a penalty for wild guesses. Draw neat and clean diagram/code where necessary.

Question No. 1	[5x3=15 Points] [Time: 15 Min.]

a. What are the problems associated with "Thinking Rationally" approach to AI? Explain.[5]

There are two main obstacles to this approach. First, it is not easy to take informal knowledge and state it in the formal terms required by logical notation, particularly when the knowledge is less than 100% certain. Second, there is a big difference between solving a problem "in principle" and solving it in practice. Even problems with just a few hundred facts can exhaust the computational resources of any computer unless it has some guidance as to which reasoning steps to try first.

b. Define what do we mean by a State-Space- Search Problem? What are its components? [5]

State Space Search Problem is a representation of a problem as a collection of different states . All the states the system can be in are represented as an abstract object(node), generally an atomic representation of the problem. An action that can change the system from one state to another (e.g. a move in a game) is represented by a link from one (object)node to another.

States: It represents a collection of all possible world states.
Initial state: Any state can be designated as the initial state.
Actions: It represents the possible rules for transition among states.
Transition model: The actions have their expected effects it is a function that maps a state and action pair to another state.
Goal test: A distinct state designated as goal.
Path cost: A possible state-transition sequence to acquire a goal state starting from an initial state.

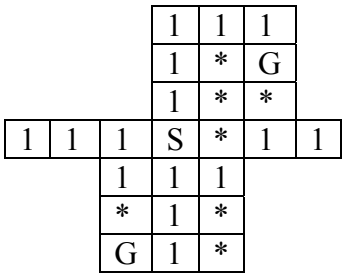
c. What are the main differences between "Learning Element" and "Performance Element" of a general Learning Agent? [5]

A learning agent can be divided into four conceptual components (i)Learning element (ii) Performance element (iii) problem generator and (iv)critic. The most important distinction is between the learning element, which is responsible for making improvements, and the performance element, which is responsible for selecting

external actions for a goal. The performance element is what considered to be the entire agent in a goal based agent/utility agent: it takes in percepts and decides on actions. The learning element uses feedback from the critic on how the agent is doing and determines how the performance element should be modified to do better in the future(performance element do change by learning element).

Question No. 2	[20 Points] [Time: 30 Min.]

Consider our Robo is standing at point(0,0) in a grid consisting of cells. Robo is standing at some start state and needs to find a path to the goal state(s) -"G". Our Robo can only move (UP, LEFT, DOWN and RIGHT)- one cell at a time, with a cost of 1. There are some cells with a blockage (*) so the Robo has to detour around these cells. Robo only need to stationed at any G. Consider the following example grid:



Each cell has a unique (i,j), considered as a node for exploration in search space.

a. If we use Breadth-first search for finding the path in the above example. Consider the frontier contains S as initial node, give the order of the cells explored if our Robo preferred to move (UP, LEFT, DOWN and RIGHT)- as preferred directions (operators/actions). [5]

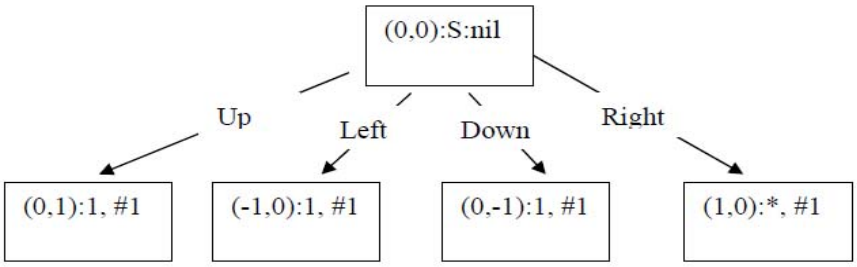
Step No 1:

In BFS, we used a Queue to hold frontier, hence when the BFS start, it will only contains Frontier ={ <(0,0:S: nil>}
we representing a state as (x,y) -position and content at the position with the path which explored this node. As it is an initial node its parent is nil.

Explored= {};

BFS first get a node from frontier, if it is not blocked one, generate all its successors (applying operators as per the preferred order of the Robo that is (UP, LEFT, DOWN and RIGHT) if applicable. To make life simple keep track of a visited state{ Note: it is a graph search problem, we need to keep track of visited node}. We can further simplify if we do not put a generated node with * as blocked cell, this solution is

putting all nodes on frontier, and only expand if possible. Current situation corresponds to the following figure:



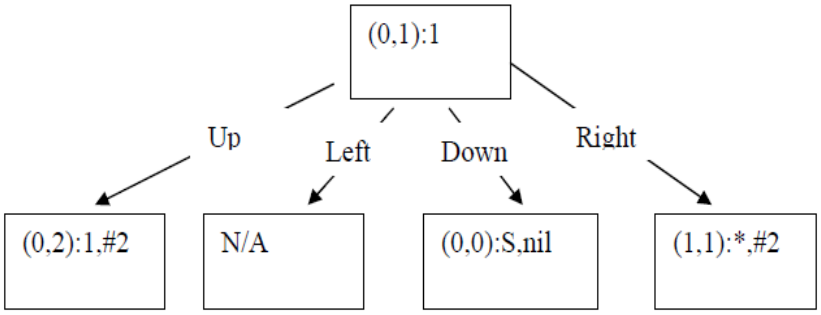
as the extracted state is not a goal state, we put it in explored set.

Step No. 2

Frontier Contains ={ <(0,1):1,#1> , <(-1,0):1,#1> , <(0,-1):1,#1> , <(1,0):*,#1>}

Explored= {<(0,0:S:nil>}

The next node that extracted from Frontier is <(0,1):1,#1> , we generate it successors



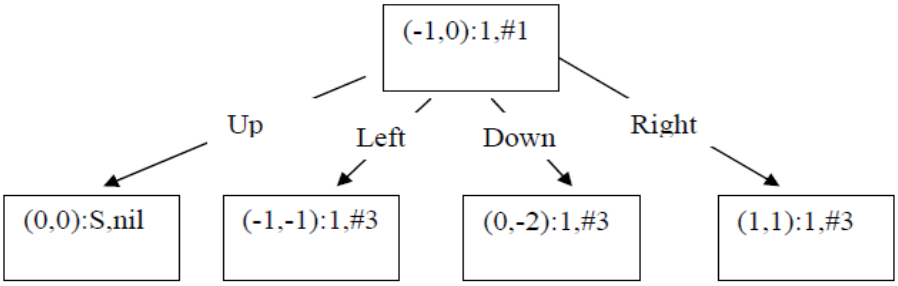
append the non-visited nodes at end of the Frontier. As the extracted state is not a goal state, we put it in explored set.

Step No. 3

Frontier Contains ={ <(-1,0):1,#1> , <(0,-1):1,#1> , <(1,0):*,#1>,<(2,0):1,#2> , <(1,1):*,#2>}

Explored= {<(0,0:S,nil>, <(0,1):1,#1> }

The next node that extracted from Frontier is <(-1,0):1,#1> , we generate it successors



append the non-visited nodes at end of the Frontier. As the extracted state is not a goal state, we put it in explored set.

Step No. 4

Frontier Contains ={ <(0,-1):1,#1> , <(1,0):*,#1>,<(2,0):1,#2> , <(1,1):*,#2>, <(-1,-1):1#3>, <(0,-2):1#3>, <(1,1);1,#3> }

Explored= {<(0,0:S,nil>, <(0,1):1,#1>, <(-1,0):1,#1> }

The next node that extracted from Frontier is <(0,-1):1> , we generate it successors, append the non-visited nodes at end of the Frontier. As the extracted state is not a goal state, we put it in explored set.

			#12	#18	#21	
			#6	#13	(2,2):G	
			#2	#7	(2,1):*	
#14	#8	#3	#1	#5	(2,0):1	(3,0):1
		#9	#4	#11		
		#15	#10	#17		
		#19	#16	#20		

Doing in this way, at step 19 we will find the goal state. The complete execution is presented in this map using step no. as # step, when we extract node <(-1,-3):G> from the Frontier, we expand it and our goal test will be true; hence he have found a solution.

Frontier ={ <(-1,-3):1,#16> ,< (1,-3):1,#16> , <(1,3):1,#18>}

Extracted Node in this step: <(-1,-3):G,#16 > no operator give new node. Goal test is true.

Frontier ={< (1,-3):1,#16> , <(1,3):1,#18>}

Explored= {<(0,0:S,nil>, <(0,1):1,#1>, <(-1,0):1,#1> , < (0,-1):1,#1>, <(1,0):*,#1>, <(0,2):1,#2>, < (1,1):,*,#2>, <(-2,0):1,#3> , <(-1,-1):1,#3>, <(0,-2):1,#4>, <(1,-1):1,#4>, <(0,3):1,#6>, <(1,2):*,#6>, <(-3,0):1,#8>, <(-1,-2):1,#9>, <(0,-3):1,#10>, <(0,-2):*,#10>, <(1,3):1,#12>, < (-1,-3):1,#16>}

The solution path is

Path= { <(-1,-3):G,#16 > , <(0,-3):1,#10>, <(0,-2):1,#4>, <(0,-1):*,#1>,<(0,0:S,nil> } This path is the optimal one as all the path between cell has equal cost.

b. If we use Depth-first search for finding the goal-path in the above example. Consider the frontier contains S as initial node, give the order of the cells explored in this case. [5]

In DFS, we used a Stack to hold frontier, hence when the DFS start, it will only contains Frontier ={ <(0,0:S: nil>}
we representing a state as (x,y) -position and content at the position with the path which explored this node. As it is an initial node its parent is nil.

Explored= {};

DFS first get a node from frontier, if it is not blocked one, generate all its successors (applying operators as per the preferred order of the Robo that is (UP, LEFT, DOWN and RIGHT) if applicable. To make life simple, ensure that the preferred operator is available on stack every time. Hence, we apply these operator in reverse order, it is purely a simplification step. To make life more simple keep track of a visited state{ Note: it is a graph search problem, we need to keep track of visited node}. We can further simplify if we do not put a generated node with * as blocked cell, this solution is putting all nodes on frontier, and only expand if possible.

Step No. 1

Frontier ={ <(0,0:S: nil>} and Explored= {};

We extract node from Frontier and expand in reverse of operator precedence , hence we get the following Frontier:

Frontier Contains ={ <(0,1):1,#1> , <(-1,0):1,#1> , <(0,-1):1,#1> , <(1,0):*,#1>}
as <(0,0:S:nil> is not our Goal state put it in explored list.

Explored= {<(0,0:S:nil>}

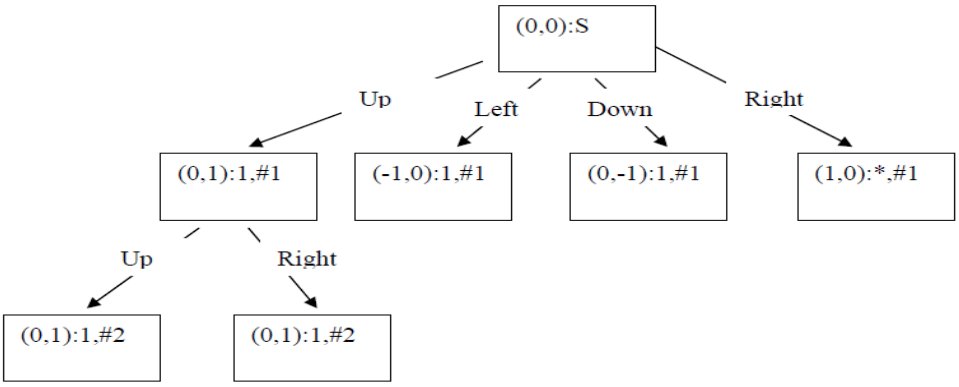
Step No. 2

We extract node= <(0,1):1,#1> from Frontier and expand in reverse of operator precedence and put it on Frontier, we will get

Frontier Contains ={ <(0,2):1,#2>, <(1,1):*#2>, <(-1,0):1,#1> , <(0,-1):1,#1> , <(1,0):*,#1>}
as <(0,1):1,#1> is not our Goal state put it in explored list.

Explored= {<(0,0:S:nil>,<(0,1):1,#1>}

Following diagram corresponds to step#2



Step No. 3

We extract node= <(0,2):1,#2>, from Frontier and expand in reverse of operator precedence and put it on Frontier, we will get

Frontier Contains ={ <(0,3):1,#3>, <(1,2):*,#3>, <(1,1):*#2>, <(-1,0):1,#1> , <(0,-1):1,#1> , <(1,0):*,#1>}

Explored= {<(0,0:S:nil>,<(0,1):1,#1>, <(0,2):1,#2>}

going this way, on step no. 7 we will get our goal state.

Path={ <(2,2):G,#6>, <(2,3):1,#5>, <(1,3):1,#4>, <(0,3):1,#3>,<(0,2):1,#2>, <(0,1):1,#1>, <(0,0):S,nil> } this solution is not optimal obviously. The solution demands the explored node order hence it is given below:

Explored = { <(0,0):S,nil>, <(0,1):1,#1>, <(0,2):1,#2>, <(0,3):1,#3>, <(1,3):1,#4>, <(2,3):1,#5>, <(2,2):G,#6>}

c. If we use Iterative Deepening search for finding the goal-path in the above example. Consider the frontier contains S as initial node, give the order of the cells explored in this case.[5]

I think you people are now in a position to write the explored order for iterative deepening. Taking solution from BFS repeat the node at each level of exploration hence:

Explored= {p1= <(0,0:S,nil>,
p2=p1, <(0,1):1,#1>,
p3= p1, <(-1,0):1,#1> ,
p4= p1, < (0,-1):1,#1>,
p5= p1, <(1,0):*,#1>,
p6= p2, <(0,2):1,#2>,
p7= p2, < (1,1):,*,#2>,
p8= p3, <(-2,0):1,#3> ,
p9=p3, <(-1,-1):1,#3>,
p10= p4, <(0,-2):1,#4>,
p11=p4, <(1,-1):1,#4>,
p12=p6, <(0,3):1,#6>,
p13=p6, <(1,2):*,#6>,
p14= p8, <(-3,0):1,#8>,
p15=p9, <(-1,-2):1,#9>,
p16=p10, <(0,-3):1,#10>,
p17=p10, <(0,-2):*,#10>,
p18=p12, <(1,3):1,#12>,
p19= p16, < (-1,-3):1,#16>}

Obvious solution is identical to BFS. here p1 is a prefix path in solution p2 and in the same fashion it is prefix with all path.

d. Give the sequence of cells explored (search path) if our Robo perform A* search in this scenario. Assume that the Manhattan distance can be used as the heuristic function- h(n) for any cell. The cost to reach a cell (i,j) = i+j that is the sum of indexes i and j. Therefore, A* uses f(n)=g(n)+h(n); [5]

In this problem you need to first assume that we are interested in optimal goal state(a single goal is required for A*). hence we can guarantee a single cost for our functions g(n) and h(n) consequently a unique cost for f(n). Starting from the initial state where our Frontier {Priority queue in this case-based on f(n)}contain S with a cost 0.

Frontier= { <(0,0):S, nil> cost=0}
Explored= {}

Step No. 1

Extract node from frontier <(0,0):S, nil> cost=0; apply applicable operators each operator can move with a cost=i+j to the cell.

For operator UP: g(n) = 0+1= 1 ; h(n)=5 hence f(n)= 6
For operator LEFT: g(n) = (-1)+0= -1 ; h(n)=3 hence f(n)= 2
For operator DOWN: g(n)=(-1)+0=-1; h(n)=4 hence f(n)=3
For operator RIGHT: g(n)=1+0=1; h(n)=5 hence f(n)=6
Put these new states with f(n) into Frontier, as <(0,0):S, nil> is not a goal state, put it on explored
hence,

Frontier Contains ={ <(-1,0):1,#1> cost 2 , <(0,-1):1,#1> cost 3 , <(0,1):1,#1> cost 6 , <(1,0):*,#1> cost 6}

Explored= {<(0,0:S:nil> cost=0}

Step No. 2

Extract node from frontier <(-1,0):1,#1> cost 2; apply applicable operators each operator can move with a cost=i+j to the cell.

For operator LEFT: g(n) = (-2)+0= -2 ; h(n)=4 hence f(n)= 2
For operator DOWN: g(n)=(-1)+-1=-2; h(n)=2 hence f(n)=0
Put these new states with f(n) into Frontier, as <(-1,0):1,#1> is not a goal state, put it on explored
hence,

Frontier Contains ={<(-1,-1):1,#2> cost 0,<(-2,0):1#2> cost 2, <(0,-1):1,#1> cost 3 , <(0,1):1,#1> cost 6 , <(1,0):*,#1> cost 6}

Explored= {<(0,0:S:nil> cost=0, <(-1,0):1,#1> cost 2, }

similarly, we more like this, on Step No. 7 we will get our goal state. The explored list will be like :

Explored= {<(0,0:S:nil> cost=0, <(-1,0):1,#1> cost 2, <(-1,-1):1,#2> cost 0, <(0,-1):1,#1> cost -3, <(0,-2):1,#2> cost -2, <(0,-3):1,#3> cost -1, <(-1,-3):G,#4 > cost 0 }

The solution path is

Path= { <(-1,-3):G,#4 > , <(0,-3):1,#3>, <(0,-2):1,#2>, <(0,-1):1,#1>,<(0,0:S,nil> }

Surprising the heuristic is neither admissible nor consistent but with a little detour we get the optimal solution.

Question No. 3	[15 Points] [Time: 15 Min.]

a. What are the drawbacks of Breadth First Search (BFS)? [3]

Breadth First Search (BFS) is generally avoided due to its high cost of time and space requirements.

b. Why Iterative Deeping Search is preferred over Breadth First Search? [3]

Iterative deepening combines the benefits of depth-first
and breadth-first search. Like depth-first search, its memory requirements are modest: O(bd) to be precise. Like breadth-first search, it is complete when the branching factor is finite. It is never preferred over BFS as it has higher cost than a BFS.

c. Illustrate how optimal A* search guarantee through a heuristic which is admissible and consistent. [3]

A* search guarantee through a heuristic which is admissible and consistent. Let we are at some initial node y, and x is our goal node. y != x . We know that f (x) ≤ f (y) by choice of x , and that |h (y) − h (x)| ≤ d (x , y) since H is admissible. Combining these two inequalities we have that g(x) ≤ g(y) + d (x , y). Since y is on the best path to x and g(y) is optimal, g(x) ≤ g(y) + d (x , y) means that g(x) is optimal.

d. Why Random Restart Hill Climbing is complete? Explain.[3]

Radom Restart Hill Climbing conducts a series of hill-climbing searches from randomly generated initial states until a goal is found.
It is trivially complete with probability approaching 1, because it will eventually generate a goal state as the initial state. If each hill-climbing search has a probability p of success, then the expected number of restarts required is 1/p for making it complete.

e. What do we mean by a one-dimension state space graph? What do we have on X-axis? What do we have on Y-axis? What are the different "geographical" landmarks that can appear on this kind of a graph? Explain each one with one line description. [3]

A one-dimensional state-space landscape is often useful in understanding local search. On X-axis it has “location” (defined by the state) and on Y-axis it has “elevation” (defined by the value of the heuristic cost function or objective function).

The graph corresponds to the different values of objective functions of different states(generally arrange in neighborhood using some applicable operators).
Local maxima: a local maximum is a peak that is higher than each of its neighboring states but lower than the global maximum.
Ridges: a ridges corresponds to a series of local maxima.
Plateaux: a plateau is a flat area of the state-space landscape from which progress may not be possible..
Shoulder: a shoulder is also a flat area from which progress is possible.
Global maxima: the highest peak—of the objective function.

<Best of Luck>