

Midterm Exam Solutions.

Artificial Intelligence (Spring 2008)

Question 1

Define in your own words the following terms:

a) **State**

A **state** is a representation that an agent can find itself in. There are two types of states: world states (the actual concrete situations in the real world) and representational states (the abstract descriptions of the real world that are used by the agent in deliberating about what to do).

b) **State space graph**

A **state space** is a graph whose nodes are the set of all states, and whose links are actions that take the agent from one state into another.

c) **Search tree**

A **search tree** is a tree (a connected graph with no undirected loops) in which the root node is the start state and the set of children for each node consists of the states reachable by taking any action.

d) **Search node**

A **search node** is a node in the search tree.

e) **Goal**

A **goal** is a state that the agent is trying to reach.

f) **Successor function**

A **successor function** described the agent's options: given a state, it returns a set of (action, state) pairs, where each state is the state reachable by taking the action.

g) **Branching factor**

The **branching factor** in a search tree is the number of actions available to the agent.

Question 2

A* is a best first search for $f(n)=g(n) +h(n)$ where $g(n)$ is the cost to reach the node n and $h(n)$ is a heuristic function from the node n to the goal. Now, given that the graph is a tree, choose special functions $g(n)$ and $h(n)$, as general as possible, that will make A* search become

a) Breadth-first search,

Breadth-first search is A* search with $h(n)=0$ and $f(n) = \text{depth}(n)$;

b) Depth-first search

Depth-first search is A* search with $h(n)=0$ and $f(n) = - \text{depth}(n)$;

c) Uniform-cost search

Uniform-cost search is A* search with $h(n)$ and so $f(n) = g(n)$.

Question 3

Suppose that an agent is in a 5 x 5 maze environment like the one shown in figure1. The agent knows that its initial location is (1,1), that the goal is at (5,5) and that the four actions, *Up*, *Down*, *Left*, *Right* have their usual effects unless blocked by a wall. Moreover, some locations have holes and should not be visited (the agent will fall to never recover). A map is given to the agent and so the agent does know where the walls are and where the holes are. The agent must arrive to the goal by the shortest path, counted as the number of sites visited. The agent uses as heuristics “the minimum number of sites needed to reach the goal, regardless of the existence of walls or holes”.

Apply the A* algorithm on a tree, starting from the state/node (1,1), as described above. Use the coordinates of the sites as the states. Build a tree search A* describing the fringe of the A* search, a list of states, as the algorithm progress from the root (node (1,1)).




4	3	2	1	Goal 0
(1,5)	(2,5)	(3,5)	(4,5)	(5,5)
5	HOLE 4 	3	HOLE 2 	1
(1,4)	(2,4)	(3,4)	(4,4)	(5,4)
6	5	4	3	2
(1,3)	(2,3)	(3,3)	(4,3)	(5,3)
7	6	5	4	3
(1,2)	(2,2)	(3,2)	(4,2)	(5,2)
Start 8	HOLE 7 	6	5	4
(1,1)	(2,1)	(3,1)	(4,1)	(5,1)

Figure 1: The maze where the agent will walk to the goal. We place the h cost, heuristic cost, at each square.

Solution:

Cost of moving one valid square (Left, Right, Up or Down): 1

Cost of moving to a hole: Infy (a very large value to make it prohibitive)

Walls do not allow moving through .

We added the heuristic value at each square into figure 1.

Cost of Start (1,1)

Expand (1,1): (1,2)→ $f=g+h=1+7=8$ (2,1)→ infy

Expand (1,2): (2,2)→ $8=2+6$, (1,1)→ $10=2+8$, (2,1)→ infy

Expand (2,2): (3,2)→ $8=3+5$, (1,2)→ $10=3+7$, (1,1)→10, (2,1)→ infy

Expand(3,2): (3,3)→ $8=4+4$, (4,2)→ $8=4+4$, (2,2)→ $10=4+6$, (3,1)→ $10=4+6$, (1,2)→10, (1,1)→10, (2,1)→ infy

Here a choice of expanding (3,3) or (4,2) is arbitrary. Let us make the worse choice (and later we will have to expand (4,2)). So if you chose to expand (4,2) at this moment, the solution is shown here by skipping the next three steps and jumping to the expansion of (4,2).

Expand (3,3): (3,4)→ $8=5+3$, (2,3)→ $10=5+5$, (3,2)→ $10=5+5$, (4,2)→ 8, (2,2)→ 10, (3,1)→ 10, (1,2)→10, (1,1)→10, (2,1)→ infy

Expand (3,4): (3,5)→ $8=6+2$, (2,4)→infy, (3,3)→ $10=6+4$, (2,3)→10, (3,2)→10, (4,2)→ 8, (2,2)→ 10, (3,1)→ 10, (1,2)→10, (1,1)→10, (2,1)→ infy

Expand (3,5): (2,5)→ $10=7+3$, (3,4)→ $10=7+3$, (2,4)→infy, (3,3)→ 10, (2,3)→10, (3,2)→10, (4,2)→ 8, (2,2)→ 10, (3,1)→ 10, (1,2)→10, (1,1)→10, (2,1)→ infy

We are back at expanding (4,2) since it is at this moment the lowest cost one, 8.

Expand (4,2): (4,3)→ $8=5+3$, (3,2)→ $10=5+5$, (4,1)→ $10=5+5$, (2,5)→10, (3,4)→10, (2,4)→infy, (3,3)→10, (2,3)→10, (3,2)→10, (4,2)→ 8, (2,2)→ 10, ((3,1)→ 10, 1,2)→10, (1,1)→10, (2,1)→ infy

Expand (4,3): (5,3)→ $8=6+2$, (4,4)→infy, (4,2)→ $10=6+4$, (3,2)→10, (4,1)→10, (2,5)→10, (3,4)→10, (2,4)→infy, (3,3)→ 10, (2,3)→10, (3,2)→10, (4,2)→ 8, (2,2)→ 10, (3,1)→ 10, (1,2)→10, (1,1)→10, (2,1)→ infy

Expand (5,3): (5,4)→ $8=7+1$, (4,3)→ $10=7+3$, (5,2)→ $10=7+3$, (4,4)→infy, (4,2)→10, (3,2)→10, (4,1)→10, (2,5)→10, (3,4)→10, (2,4)→infy, (3,3)→ 10, (2,3)→10, (3,2)→10, (4,2)→ 8, (2,2)→ 10, (3,1)→ 10, (1,2)→10, (1,1)→10, (2,1)→ infy

Expand (5,4): (5,5)→ $8=8+0$, (4,4)→infy, (5,3)→ $10=8+2$, (4,3)→10, (5,2)→10, (4,4)→infy, (4,2)→10, (3,2)→10, (4,1)→10, (2,5)→10, (3,4)→10, (2,4)→infy, (3,3)→ 10, (2,3)→10, (3,2)→10, (4,2)→ 8, (2,2)→ 10, (3,1)→ 10, (1,2)→10, (1,1)→10, (2,1)→ infy

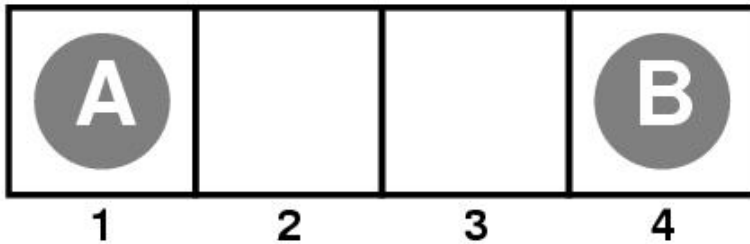
Expand (5,5) ...Goal! finished.

Question 4

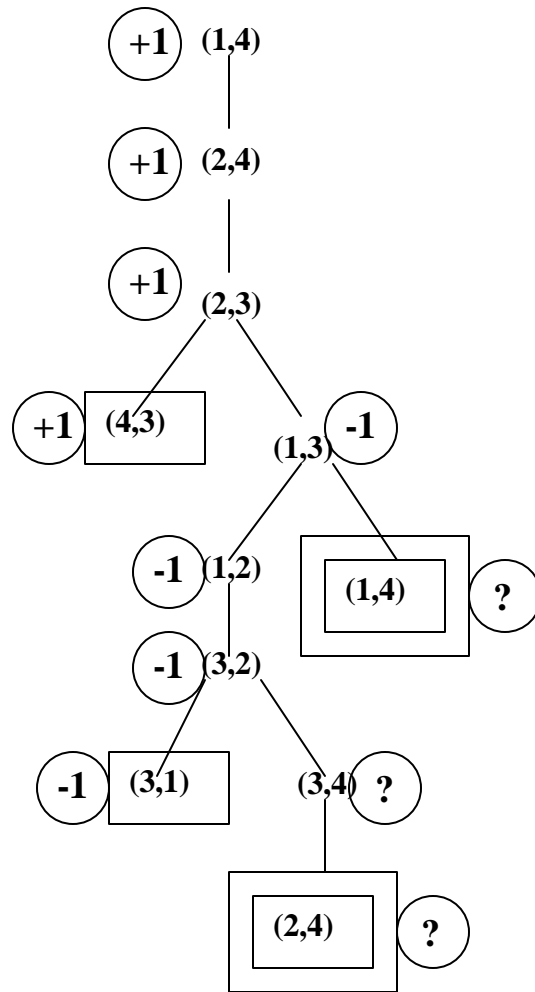
Consider the two-player game described below (Figure 1).

- a) Draw the complete game tree, using the following conventions:
- Write each state as (s_A, s_B) where s_A and s_B denote the token locations.
 - Put each terminal state in a square boxes and write its game value in a circle next to it.
 - Put *loop states* (states that already appear on the path to the root) in double square boxes. Since it is not clear how to assign values to loop states, annotate each with a “?” in a circle.
- b) Now mark each node with its backed-up minimax value (also in a circle). Explain how you handle the “?” values and why.

Figure 1:



This is the starting position of a simple game. Player A moves first. The two players take turns moving, and each player must move his token to an open adjacent space *in either direction*. If the opponent occupies an adjacent space, then a player may jump over the opponent to the next open space if any (for example, if A is on 3 and B is on 2, then A may move back to 1). The game ends when one player reaches the opposite end of the board. If player A reaches space 4 first, then the value of the game to A is +1; if player B reaches space 1 first, then the value of the game to A is -1.



The game tree for the four-square game. Terminal states are in single boxes, loop states in double boxes. Each state is annotated with its minimax value in a circle.

The “?” values are handled by assuming that an agent with a choice between winning the game and entering a “?” state will always choose the win. That is, $\min(-1, ?)$ is -1 and $\max(+1, ?)$ is $+1$. If all successors are “?”, the backed-up value is “?”.