ECE469 – Pset 1

Jonathan Lam

October 5, 2020

1. *For each task environment, indicate what categories the task environment belongs to. Also state whether the rational agent would base its decisions on only the current percept or the entire percept history.*

   (a) *A program that plays Go against the user.*

fully observable	entire Go board is observable
strategic	the current state is based on your past moves and your opponent's move (which you cannot control)
sequential	previous moves affect current available moves
semi-dynamic	there may be time limits on a move, but otherwise nothing changes while you're thinking about the next move to make
discrete	finite, non-continuous possible next moves
multi-agent	a second player is playing against you

   All of the possible moves are governed by the current board state, so the program need only look at the current board state to determine a next move. (This is as far as I can tell by briefly skimming over the rules, but I am not sure if there are any moves like a castle that do indeed depend on previous state and are not knowable from the current board state.)

   (b) *A program controlling a robotic arm that detects and stacks Jenga pieces (one of the senior projects this year!). The goal is not to have the robotic arm play the game, but rather it will detect pieces on a surface and build the initial tower.*

   The assumption here is that this robot is in an generic uncontrolled setting, where other factors may interfere and the Jenga blocks may not be entirely uniform. We can consider any unexpected behavior (e.g., bug landing on block, wind gust, or person moving block) to be part of a stochastic environment, rather than an adversarial agent.

partially observable	sensors may not capture every minute aspect of the Jenga tower and other environmental conditions (e.g., temperature, humidity)
stochastic	other unspecified environmental factors (e.g., a person or another robot nearby) may affect the environment
sequential	previous moves affect the shape of the tower (and thus the next moves)
dynamic	the tower can fall over or be otherwise altered while the program is deciding which move to make
continuous	infinitely many different actions (even if there are finitely-many pieces, any piece can be moved in infinitely many different ways)
single agent	no other specified agents

What information the robot uses depends on how it calculates the next move. If it creates an internal physical model of the world which it updates periodically with its sensors (which is common in robots; credit to Derek Lee for suggesting this idea to me, since I don't know much about robotics), then it relies on this model (and thus its percept history). However, it is also possible that a simpler robot simply takes a snapshot of the current situation with its sensors and tries to choose the next move based only on its current percept.

(c) *A face recognition system that detects faces in an image and looks them up in a database of known faces to find matches (assume the system has already been trained).*

fully observable	entire image is observable
deterministic	there is no random actions happening in an episode; the algorithm proceeds deterministically given an image
episodic	current image recognition doesn't depend on previous images (this would be different if it were training on this dataset)
static	image and database doesn't change while the system is processing an image
continuous	using the given assumption that an image is continuous
single agent	no other agents

Since the system is already trained, it only depends on the current percept (the current image) to determine whether the face matches something in the dataset. (I.e., to say that it uses its percept history means that it uses other images from the evaluation dataset to inform its opinions, which is false.)

2. *Assume a program is playing a deterministic (really strategic), turn-taking, two-player, zero-sum game of perfect information.*

(a) *Assuming that no pruning occurs, fill in the minimax values of all the internal nodes (including the root) in the game tree. Also circle the move that is selected according to the search.*

See Figure 1.

(b) *Now assuming that alpha-beta pruning occurs, draw lines through the edges in the game tree that separate the nodes at which the pruning decision occurs from the nodes which are never considered.*

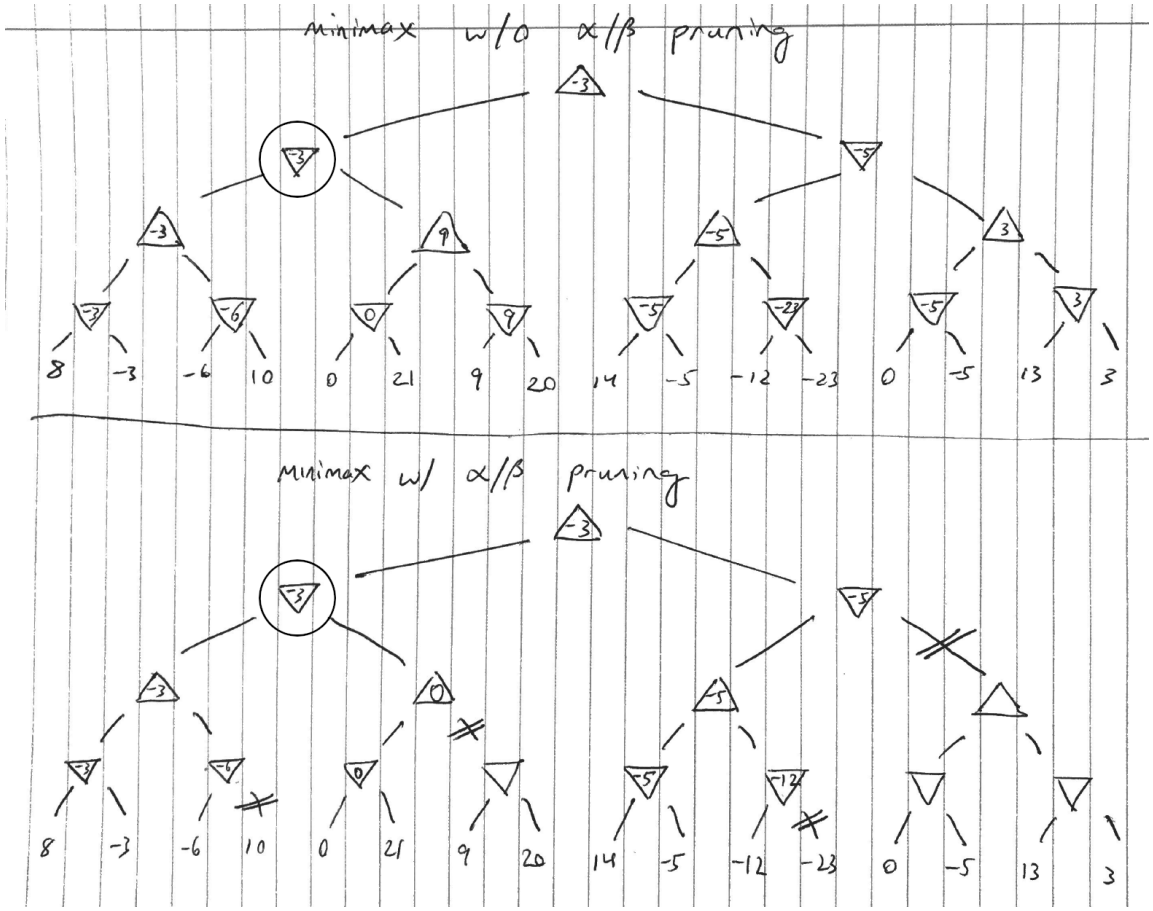


Figure 1: Minimax search, with and without α-β pruning. The circled node is the selected next action for MAX. It is clear that both give the same minimax values.

I did the pruning intuitively, so the α and β values are not shown. It is clear that the minimax values at each of the nodes where it is generated is equal to the unpruned tree. The intuition for each prune, from left to right:

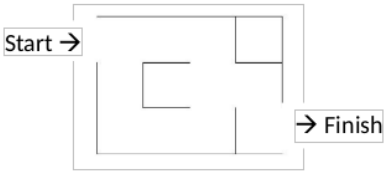
- -6 at the MIN node was lower than the -3 value at its parent MAX node, so it wouldn't be chosen by MAX at the parent node
- 0 at the MAX node is higher than the -3 value at its parent MIN node, so it wouldn't be chosen by MIN at the parent node
- -12 at the MIN node is lower than the -3 value at the root MAX node, so it wouldn't be chosen by MAX at the root
- -5 at the MIN node is lower than the -3 value at the root (its parent) MAX node, so it wouldn't be chosen by MAX at the root

(c) *If both players play perfectly, what will be the outcome of the game (i.e., will MAX or*

MIN win or will it be a draw)? Briefly explain your answer.

MIN will win with a score of (-)3, as evident by the value of the root node. In other words, this is the minimax value, which is the end result given considering the worst case scenario for both players (if both players played optimally).

3. *Consider an AI software agent (a program) that processes a maze contained in an $N \times M$ grid, where N (the number of rows) and M (the number of columns) are guaranteed to be at least 2. The entire maze is available to the agent at once, not as an image, but as an array indicating where the walls are. The agent is tasked with finding a path from the top left square to the bottom right square. The path cost is the number of steps (i.e., every move from one square to an adjacent square adds one to the cost). Only horizontal and vertical steps are allowed. Walls will only occur between squares. It is possible that there may be loops, multiple paths to the solution, or perhaps no path to the solution. An example of such a maze, where N is 3 and M is 4:*



   (a) *First consider applying a tree search version of depth-first search (DFS) to this problem. For this question, consider a version that does not remember all reached nodes, but that does remember nodes on the current path to avoid cycles. Is this strategy complete? Is this strategy optimal?*

   This strategy is complete (remembering nodes on the current path will avoid infinite loops, and DFS will eventually expand all nodes in the (finite) space just like BFS will), but not optimal (the first solution found will not necessarily be the shortest path to the bottom right, it will just be *a* path to the bottom right).

   (b) *Now consider applying a graph search version of breadth-first search (BFS) to this problem, as discussed in class. Is such a strategy complete? Is such a strategy optimal?*

   Since the search space is a finite board (and we can easily keep track of repeated nodes since this is a graph-search algorithm), this is complete. BFS is also optimal since we are looking for the shortest path to the finish, which BFS will find first.

   (c) *Now consider applying the graph search version of A* search, as discussed in class, to the problem. You will consider four heuristic, where x is the current row and y is the current column. Assume rows are labeled from 1 to N and columns are labeled from 1 to M; the top-left cell (the starting position) is $(1, 1)$, and the bottom-right cell (the destination) is (N, M). The four heuristics we are considering are:*

$$H_1(n) = 0$$
$$H_2(n) = N - x + M - y$$
$$H_3(n) = \sqrt{(N-x)^2 + (M-y)^2}$$
$$H_4(n) = (N - x)(M - y)$$

Which of the above four heuristics would guarantee that the graph search version of A search is optimal? Specify all that apply.*

Any admissible heuristic will lead to an optimal A* search. In particular, the smallest number of moves between two points when only horizontal or vertical moves are considered is the Manhattan distance. Thus, any of the heuristics which is guaranteed to be no greater than the Manhattan distance will guarantee optimality with A*. In particular:

- $H_2 \geq H_1 = 0$ since $N > x$ and $M > y$ (i.e., Manhattan distance is positive).
- $H_2 \geq H_3$ because of the triangle inequality.
- H_2 is not always greater than H_4 (take for instance when $N - x = 2$ and $M - y = 3$).

H_4 is thus not admissible, and it can be shown to be not optimal by a counterexample (in which $H_4 >$ Manhattan distance for some tile on the optimal path, and thus the optimal path is not chosen by A*).

(d) *Which of the above heuristics is the best one to use with the graph search version of A* search? Why?*

In general, the largest admissible heuristic is the most efficient, since it gets us closer to the actual total path costs. H_2 (the Manhattan distance) dominates the other admissible heuristics by the arguments in the previous answer (in fact, it is the largest admissible heuristic for this problem), so it is the best one to use with A*.