Assignment 2 Report

Saurabh Daptardar, Srikanth Kuthuru

September 21, 2017

1 Problem 1:

a. The solution is 3. 'A' should go towards 'C'. This maximizes the final score it can get, i.e. 3

b. Using alpha-beta pruning and standard left-to-right evaluation will end up checking 12 leaf nodes.Refer to Figure 1 for the graph.

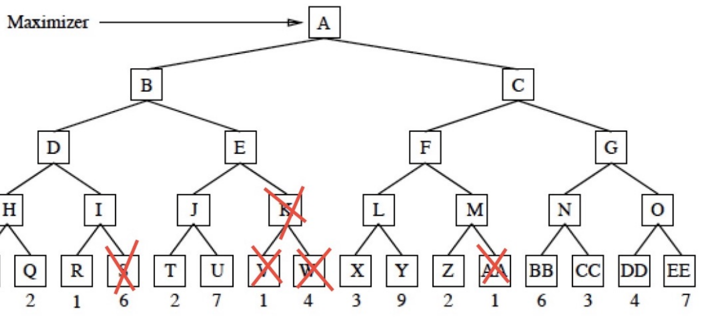


Figure 1: 1b

c. For the right-to-left evaluation, we check 15 leaf nodes. Here, we end up up checking more nodes when compared to the left-to-right evaluation. Refer to Figure 2 for the graph.

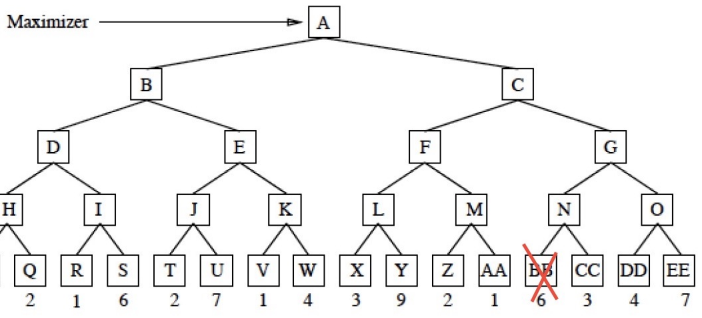


Figure 2: 1c

d. A heuristic function has to be designed. At any node N, the heuristic function $h(.)$ should give a value $h(N, A)$ where A is any possible action at that node. If N is max node, then search the branch towards an action that has maximum heuristic value. Likewise, if N is a min node, then search the branch towards an action that has minimum heuristic value.

As an example, for the given graph assume that we had the perfect heuristic function. This heuristic function can tell the actual value that a edge has when Minimax is used. Then we end up searching only 11 nodes. Refer to Figure 3 for the graph.

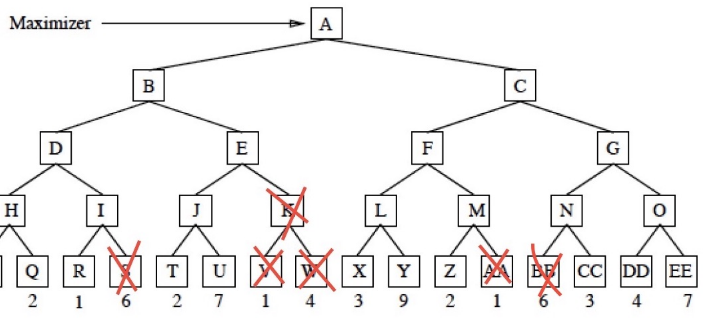


Figure 3: 1d

e. Game play searches do not start from end state because, there are large number of possible end states. Searching all of them would be computationally expensive.

2 Problem 2:

a. Consider any game tree, assuming all the nodes have been evaluated till depth 1. For example refer Figure 4. Let the root node be maximizer and the next level be either

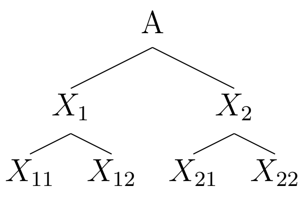


Figure 4: Example game tree

expectation or minimum over evaluated leaf nodes. The values of nodes at level 1 be X_i, where i is index for all actions of player 1 and leaf nodes be X_{ij}, for parent X_i, where j, is index for all actions of opponent

For expectation case, $X_i = \mathbb{E}_j[X_{ij}]$ and value at root $= \max_i(\mathbb{E}_j[X_{ij}])$

For minimum case, $X_i = \min_j(X_{ij})$ and value at root $= \max_i(\min_j(X_{ij})$

We know, $\mathbb{E}_j[X_{ij}] \geq \min_j(X_{ij}) \forall i, j \implies \max_i(\mathbb{E}_j[X_{ij}]) \geq \max_i(\min_j(X_{ij}))$ so the value for expectimax is always greater than that of minimax

b. Proof above shows value at root node will always be smaller for minimax than that for expectimax

c. Player 1 should use minimax, assuming the game is zero sum and player 2 is optimal (rational) player and is trying to win.

d. Player 1 should use expectimax, assuming the opponent is random and playing randomly (like by coin toss or rolling die for example) and any outcome of player 2's play is by chance.

e. We should propagate 2 values at each node. When it's player 2's turn we should store the tuple (expected value, maximum value). Now player 2 will minimize the expected value, thereby returning tuple with minimum expected value (Emin, Max). When it's player 1's turn, player 1 should maximize over the maximum value of the tuple returned by player 2. Refer to Figure 5 for better understanding. It gives better return than what minimax would give (note: for the example given, minimax gives a value of 6, whereas this gets 50).

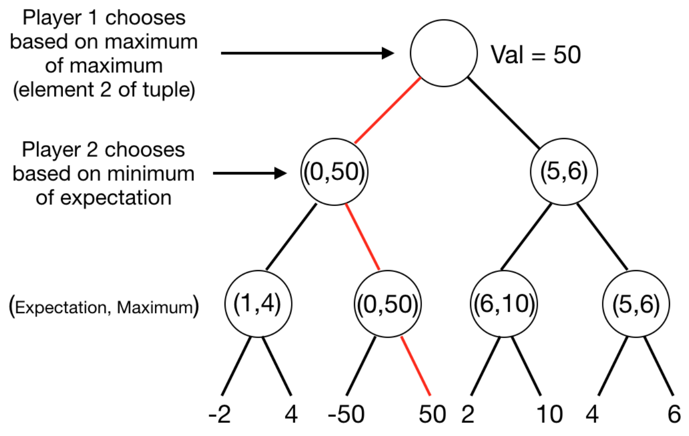

Player 1 chooses based on maximum of maximum (element 2 of tuple)
Val = 50
Player 2 chooses based on minimum of expectation
(0,50)
(5,6)
(Expectation, Maximum)
(1,4)
(0,50)
(6,10)
(5,6)
-2
4
-50
50
2
10
4
6

Figure 5: 2 e

3 Problem 3:

3.1 Warmup: Running and Understanding the code

3.2 Minimax agent value function:

Recurrence equation for *Pacman* with multiple adversaries. Here, player a_0 is the *Pacman* and $a_1, a_2, ..., a_n$ are the ghosts.

$$V_{opt}(s,d) = \begin{cases} Utility(s),\ if\ isEnd(s) = True \\ EvalFunction(s),\ if\ d = 0 \\ \underset{a \in Actions(s)}{max} V_{opt}(succ(s,a),d),\ if\ Player(s) = a_0 \\ \underset{a \in Actions(s)}{min} V_{opt}(succ(s,a),d),\ if\ Player(s) = a_1, a_2, ..., a_{n-1} \\ \underset{a \in Actions(s)}{min} V_{opt}(succ(s,a),d-1),\ if\ Player(s) = a_n \end{cases}$$

Why does the Pacman thrash around right next to a dot?
This depends on the evaluation function. When the value functions $V_{opt}(s,d)$ for all possible actions is the same, then it chooses directions randomly. It doesn't care if the food is there nearby or not.

3.3 Trapped Classic layout: Comparing Minimax and Expectimax [3.2,3.4]

In Minimax case, it chooses to die soon because it thinks that the ghosts are coming towards it. So, if it goes West it loses points because of extra steps and no food pellets to gain points, but if it goes East it can die soon while maintaining a relatively high score (-501) when compared to going west (-502) and dying.

In Expectimax case, Pacman knows that the ghosts move randomly. So, if it goes west it has a chance to gain food pellets and if it goes east it might die. So, it always chooses West. We have compared these two agents on a trapped classic layout. Minimax agent wins 0 times while expectimax agent wins few times(around half the games).

3.4 Expectimax agent value function:

In this case, ghosts behave randomly. Therefore, we used expectimax search.

$$V_{opt}(s,d) = \begin{cases} Utility(s),\ if\ isEnd(s) = True \\ EvalFunction(s),\ if\ d = 0 \\ \underset{a \in Actions(s)}{max} V_{opt}(succ(s,a),d),\ if\ Player(s) = a_0 \\ \sum_{a \in Actions(s)} \prod(s,a) V_{opt}(succ(s,a),d),\ if\ Player(s) = a_1, a_2, ..., a_{n-1} \\ \sum_{a \in Actions(s)} \prod(s,a) V_{opt}(succ(s,a),d-1),\ if\ Player(s) = a_n \end{cases}$$

3.5 Better Evaluation Function:

We took linear combination of different aspects like distance from food pellets, ghosts and capsules. If the pacman is far from the ghosts then the state has high score. Also, if it is closer to food pellets or the capsules, the state should have high score. Eating capsules is more beneficial than pellets because the ghosts get scared and pacman can move freely. Also, eating ghosts in scared state hugely increases the final score. So, if scared ghosts are close then the state has high score, this makes the pacman move towards teh scared ghost. Inverse of distance is used to give score. Food pellets distance has a positive weight of 1. Capsule have +25, ghosts have -25, and scared ghosts have a weight of +50. This linear combination is added to the game score at that state to get the final Evaluation function value. Points are deducted for number for food pellets and capsules remaining to encourage the pacman to eat more. There is additional +50 points for finishing all food pellets.

4 Problem 4:

a. Max Trees contain only max nodes at each layer. Pruning is not possible in a max tree whose leaf values are unbounded. Since we are looking for a maximum value at every node, we cannot leave any leaf value un-searched because it might contain the maximum value in that branch.

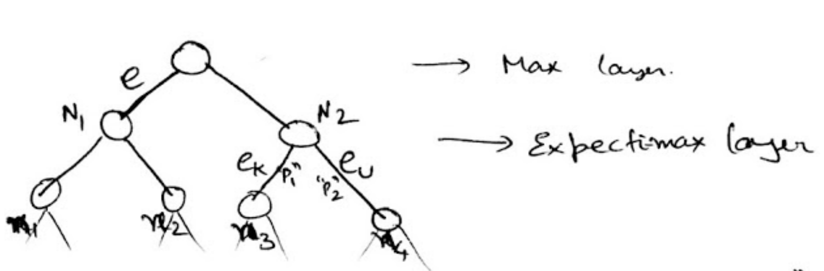
b. Using the same reasoning as above, where the leaf values are unbounded, pruning is not possible even with *expectimax* trees (given the all probability values are greater than 0). The probabilities should be $p > 0$, because if it is 0, then that branch need not be searched because we know that its output is always 0.

c. The leaf node values are bounded in $[0, 1]$. For a max tree, pruning is possible in the following case. If one of the edges at a node already has the maximum possible value (here, it is 1), then searching the remaining branches is unnecessary. This is because the max possible value the other branches can give is 1 and the max node result will still be 1.

d. Yes, pruning is possible in this case. An example is given in Figure 6.

   (Note: Figure on next page)

e. Highest probability first procedure may result in more pruning opportunities. Consider the example given before. If $p1 \gg p2$, and assume e_k was small. Then, e_u should be very high for pruning not to be done. That is, e_u should be high enough for $p_1e_k + p_2e_u > e$ to happen. So, the chances of this equation being satisfied for e_u values less 1 becomes low. Therefore, pruning chances are more.



- Assume, The left branch has been searched and the value "e" ~~was~~ is calculated.

- ~~leaf~~ n_3 node is searched and e_K is calculated.

  e_K: here "k" stands for "known".

  e_U: here "U" stands for "Unknown".

- ~~leaf~~ n_4 node is still not searched.

- At N_2 node, let the probabilities be p_1, p_2.

  So, output will be $p_1 e_K + p_2 e_U$.

- We know that the max value e_U can take is 1.

- So, if $p_1 e_K + p_2 (1) < e$, then searching "n_4" is unnecessary.

Figure 6: Expectimax Pruning