CS 217: Artificial Intelligence and Machine Learning **Jan-Apr 2024**

Lecture 20: Sequential and Simultaneous Move Games

Lecturer: Swaprava Nath *Scribe(s): SG39, SG40*

20.1 Limitations of Backward Induction and Various Enhancements

The computation of the function $u_{agent}(s)$ for all the states s that occur in the game is computationally expensive for large games like Chess which have about 10^{40}, 10^{170} nodes in the game tree. Hence solving a game exactly is not practical at all for large games. Now we see possible alternatives for enhancements like the **Depth Limited Search** and $\boldsymbol{\alpha, \beta}$ **Pruning**.

20.1.1 Depth Limited Search

Depth-limited search is a variant of depth-first search (DFS) algorithm used in graph traversal. In DFS, the algorithm explores as far as possible along each branch before backtracking. However, this approach might lead to very deep paths in some game trees.

Depth-limited search addresses this issue by imposing a depth limit on the exploration. When the algorithm reaches the specified depth limit, it stops exploring that branch further and backtracks to explore other branches. This prevents the algorithm from getting stuck in deep paths and allows it to handle very deep trees effectively thus reducing the computational time.

The utility of the agent can be written as

$$u_{agent}(s,d) = \begin{cases} utility(s) & \text{if } isEnd(s) \\ eval(s) & \text{if } d = 0 \\ \max\limits_{a \in actions(s)} \{u_{agent}(succ(s,a), d-1)\} & \text{if } player(s) = agent \\ \min\limits_{a \in actions(s)} \{u_{agent}(succ(s,a), d-1)\} & \text{if } player(s) = opponent \end{cases} \tag{20.1}$$

Here $eval(s)$ is a domain specific function denoting the possible utility to the agent. For example in the game of chess, it can be written as

$$eval(s) = army + mobility + king's\ safety + \dots \tag{20.2}$$

The contribution of army towards the utility can be represented by the following equation

$$army = 10^{100}(K - K') + q(Q - Q') + r(R - R') + \dots \tag{20.3}$$

Here K, Q, R and K', Q', R' are the number of king, queen and rooks that the agent and the opponent respectively have. The constants can be appropriately chosen according to the value of the pieces in the game. The constant 10^{100} (very large) for the kings denotes that if the agent's king is out of the board while opponent's king is still on board, then the utility is minimum for the agent.

The mobility can be expressed in the following way

$$mobility = c \times (\#legal\ moves - \#legal\ moves') \tag{20.4}$$

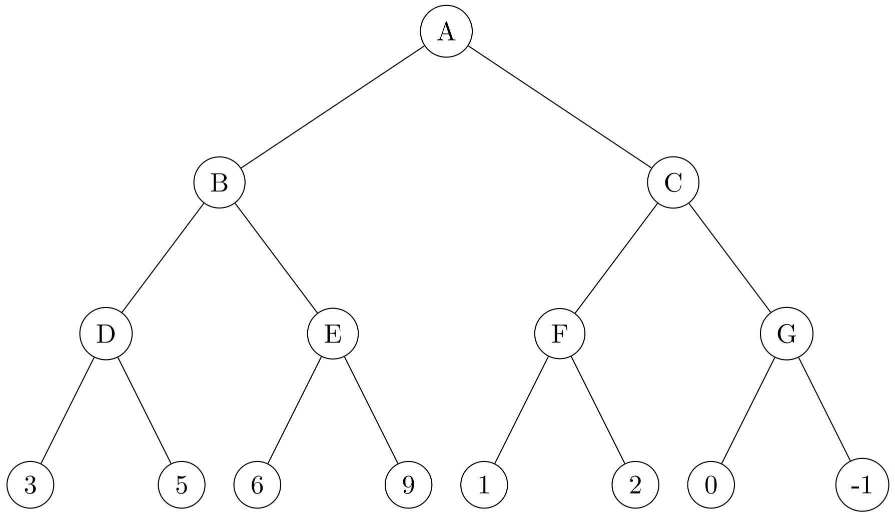
Where c is a suitable constant.

20.1.2 α, β Pruning

$\alpha, \beta - Pruning$ is a technique used in game theory to reduce the number of nodes evaluated in the minimax algorithm search tree. It aims to eliminate branches that are deemed irrelevant to the final result.

20.1.2.1 Binary Tree of Actions with Alpha-Beta Values

Consider the following binary tree representing actions in a game, along with alpha-beta values:



Here, we initialise α with -∞ and β with $+\infty$ in all action nodes.

20.1.2.2 Marking Subtrees for Pruning

Using alpha-beta pruning, we mark subtrees to be pruned based on the values of α and β. If $\alpha \geqslant \beta$ at a node, we can prune the subtree rooted at that node as it won't affect the final decision.

20.1.2.3 Formulae for Alpha and Beta

In alpha-beta pruning:

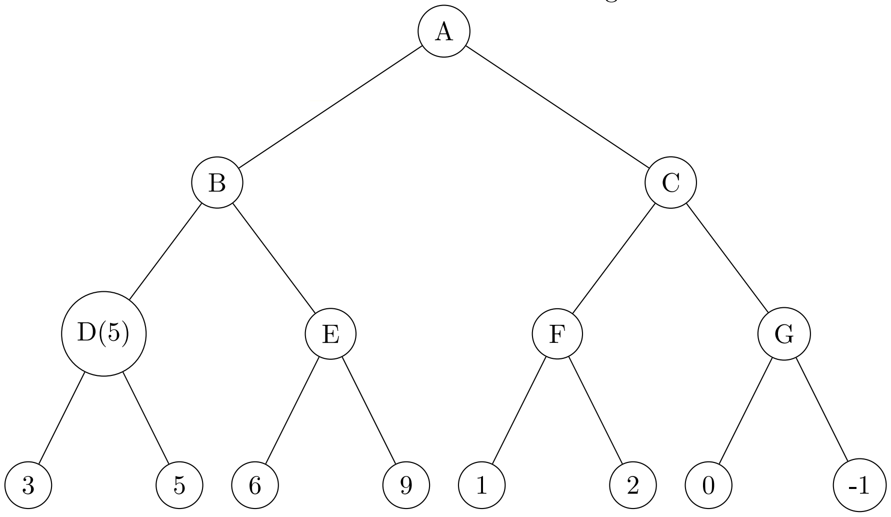
- For 'Max' nodes: $\alpha = \max(\alpha, \text{value})$
- For 'Min' nodes: $\beta = \min(\beta, \text{value})$

These updates help narrow down the search space, making game-playing algorithms more efficient.

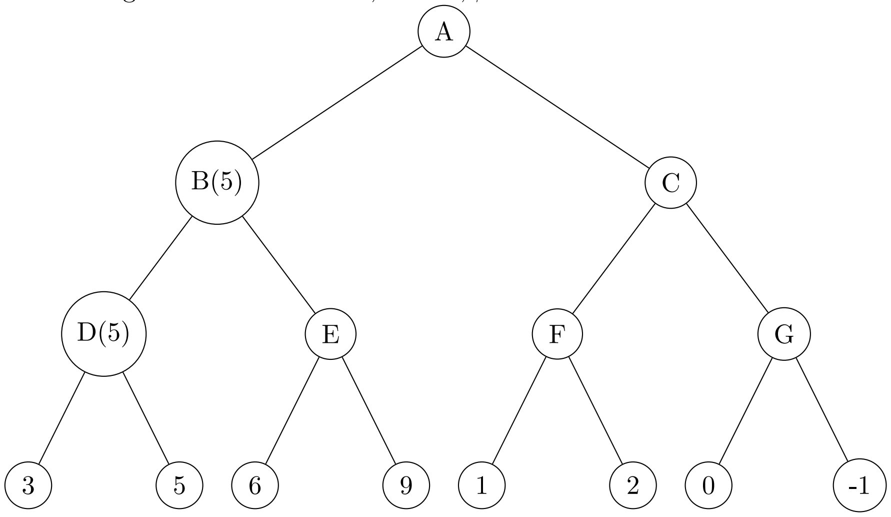
20.1.2.4 Run of the Algorithm for an Example

If, for example, at node 'Z', $\alpha \geqslant \beta$. We can then mark the subtree rooted at 'Z' for pruning. Considering the given binary tree, these are the following steps :-

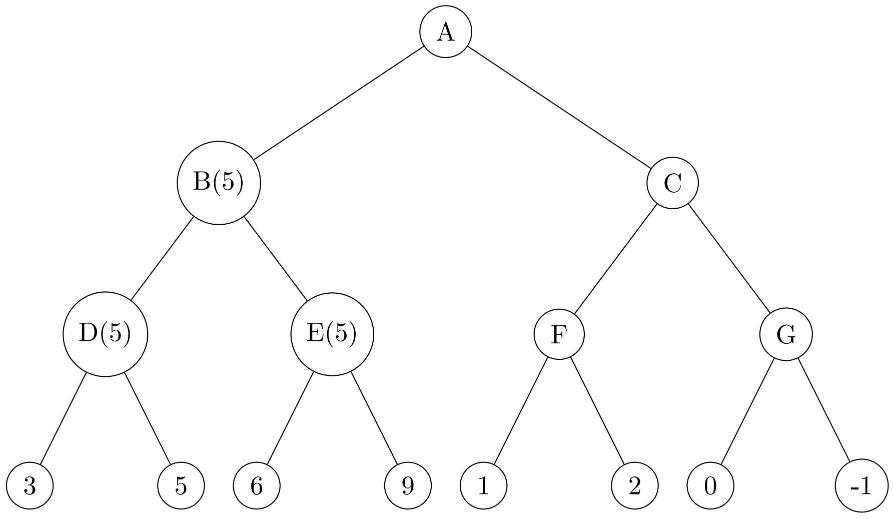
1. Start with the subtree rooted at D. After transfering values from its two leaves, α=5, β=∞.



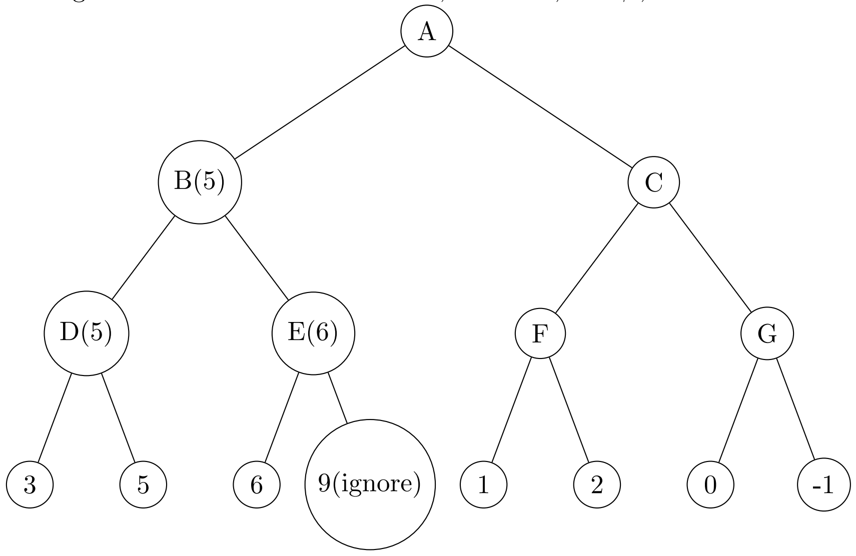
2. Transfering the value of D to B, α=-∞, β=5.



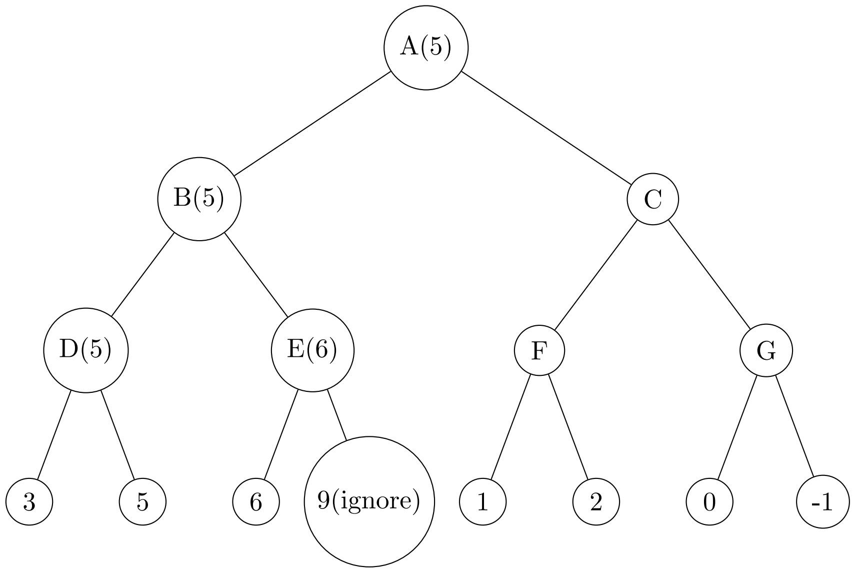
3. Transfering the value of B to E, α=-∞, β=5.



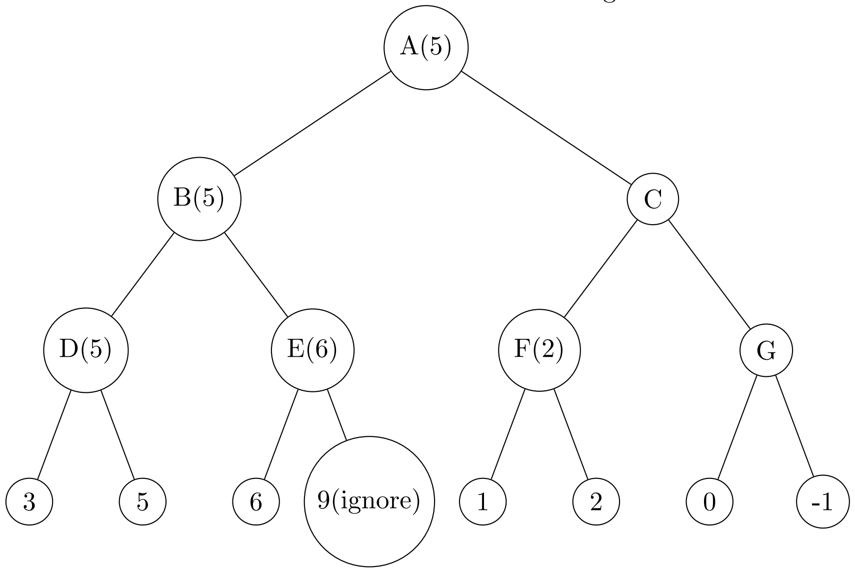
4. Taking the value from first child of E, α=6. So, $\alpha > \beta$, hence the rest branch of E is ignored.



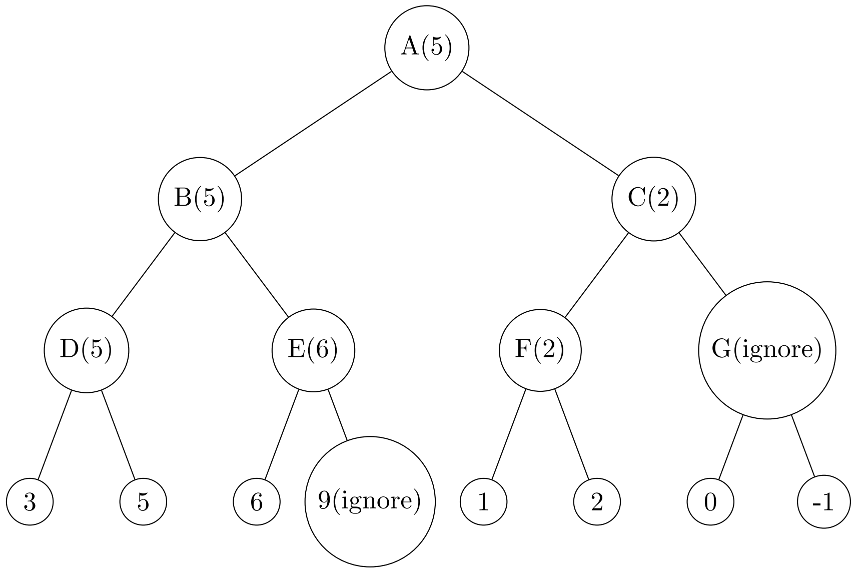
5. After transfering values from B to A, α=5, β=∞.



6. Start with the subtree rooted at F. After transfering values from its two leaves, α=2, β=∞.



7. After transfering values from A and F to C, α=5, β=2. So, $\alpha > \beta$. Hence, G subtree is ignored.



Likewise, the whole game can be solved in a computionally less expensive way by prunning.

20.2 Simultaneous Move Games

The games we have seen so far in this lecture and the previous were sequential or turn based games ie, games where players make moves in some order. However some games are simultaneous where the players might make moves at the same time. A real life example we take is foot ball. There is a shooter who is going to kick the ball and there is a goal keeper who defends the goal. Now, for simplicity assume these are the only two players playing the game. The goal keeper cannot decide which direction to move to defend the goal after the shooter shoots and has to predict and already start moving in the direction he chooses. Similarly the shooter cannot kick the ball assuming the goal keeper is going to remain where he is currently. He has to make a guess and kick it towards some portion of the goal. Again to keep things simple, we will only consider three moves for each of the two players : L, C, R denoting left,centre and right respectively.

We now define a utility function through a matrix. Let the agent be the shooter and the goal keeper the opponent. Now we define the various actions,the outcomes of every pair of actions and the rules:

1. If both players make the same move ie (L, L),(C, C) or (R, R), then, the goal keeper defends the goal successfully, and the utility matrix $U[i][i] = -1$ where $i = 0, 1, 2$ correspond to the actions L, C, R respectively.

2. $U[i][j]$ corresponds to : Player 1 taking the action i and player 2 taking the action j.

3. For any of the other 6 tuples, $U[i][j] = 1$ where $i \neq j$.

This is an example of a **2-player zero sum simultaneous move game** and is called a **Matrix Game**. Now we consider a variant of this game where the agent is an excellent left shooter ie, even if the goal keeper moves to the left, the shooter's goal is successful. So the utility matrix for the variant will have $U[0][0] = 1$

and the remaining entries are the same as before. Call the variant's matrix as U_2 and the original as U_1. Therefore we have

$$U_1 = \begin{bmatrix} -1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix} \tag{20.5}$$

$$U_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix} \tag{20.6}$$

Now we define a few more terms for matrix games :

Definition 20.1 *Equilibrium : A tuple of actions from which no player* ***gains*** *(a strict gain) by a* ***unilateral deviation****. A unilateral deviation from a tuple of actions $(a_1, a_2, \ldots, a_n)$ is a tuple of actions of the form $(b_1, b_2, \ldots, b_n)$ where $a_i = b_i$, $\forall i \neq j$ for some j ie, the actions of exactly one player j differs from the previous set.*

In our case, (L, L) is a simultaneous move equilibrium for game 2 because, the agent (player 1) will not gain any extra utility from (C, L) or (R, L) and similarly the opponent (player 2) will not gain any extra utility from (L, C) or (L, R).

However, observe that for game 1, no such equilibrium exists. This can be shown by analysing all possible pairs of actions.

1. (L,L) : Player 1 gains additonal utility by (C, L) or (R, L). Similarly for (C, C) and (R, R), player 1 stands to gain by unilaterally deviating.

2. (L,R) : Player 2 gains additonal utility by (L, L). Similarly for any tuple (A, B), where $A \neq B$, if player 2 deviates and chooses the action A, there is a utility gain.

Therefore there exists no such equilibrium for game 1.

We will now try to generalise this observation. Which types of simulataneous move games have an equilibrium?

The agent is a max player and the opponent is a min player. Therefore, the agent tries to maximise his utility at each step and the opponent tries to minimise the agent's utility thereby maximising his own utility. We compute the terms max of min and min of max for the utility matrix as follows :

1. For every row of the matrix calculate the minimum utility possible. The value corresponding to row i denotes the least utility for the agent upon performing move i. Now take the max of the min values over all rows. This quantity is denoted as $\max_{s1} \min_{s2} u(s1, s2)$.

2. Similarly, for every column of the matrix calculate the maximum utility possible. The values corresponding to column j denotes the most utility the agent can obtain if the opponent performs move j. Now take the min of the max values over all columns. This quantity is denoted as $\min_{s2} \max_{s1} u(s1, s2)$.

Lemma 20.2 $\max_{s_1} \min_{s_2} u(s_1, s_2) \leqslant \min_{s_2} \max_{s_1} u(s_1, s_2)$.

Proof: *For any* $(s1, s2)$*,*

$$\begin{aligned} u(s_1, s_2) &\leqslant \max_{s_1} u(s_1, s_2) \text{ and} \\ \min_{s_2} u(s_1, s_2) &\leqslant u(s_1, s_2) \\ \Rightarrow \min_{s_2} u(s_1, s_2) &\leqslant \max_{s_1} u(s_1, s_2) \end{aligned} \tag{20.7}$$

Since the previous inequality is true $\forall s_1$*, it is also true for* s_1^* *which maximises* $\min_{s_2} u(s_1, s_2)$*. Therefore we have,*

$$\begin{aligned} \min_{s_2} u(s_1^*, s_2) &\leqslant \max_{s_1} u(s_1, s_2) \\ \Rightarrow \max_{s_1} \min_{s_2} u(s_1, s_2) &\leqslant \max_{s_1} u(s_1, s_2) \end{aligned}$$

Similarly, since the previous inequality is true $\forall s_2$*, it is also true for* s_2^* *which minimises* $\max_{s_1} u(s_1, s_2)$*. Therefore we have,*

$$\begin{aligned} \max_{s_1} \min_{s_2} u(s_1, s_2) &\leqslant \max_{s_1} u(s_1, s_2^*) \\ \Rightarrow \max_{s_1} \min_{s_2} u(s_1, s_2) &\leqslant \min_{s_2} \max_{s_1} u(s_1, s_2) \end{aligned}$$

And we are done with the proof. ■

If for some game, these two quantities equal each other, we have an equilibrium and the equilibrium point is called the saddle point. We will continue with this concept in the next lecture.