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

Lecture 19: Two Player Competitive Games

Lecturer: Swaprava Nath *Scribe(s): SG38*

Disclaimer: *These notes aggregate content from several texts and have not been subjected to the usual scrutiny deserved by formal publications. If you find errors, please bring to the notice of the Instructor.*

19.1 Two Player Competitive Game

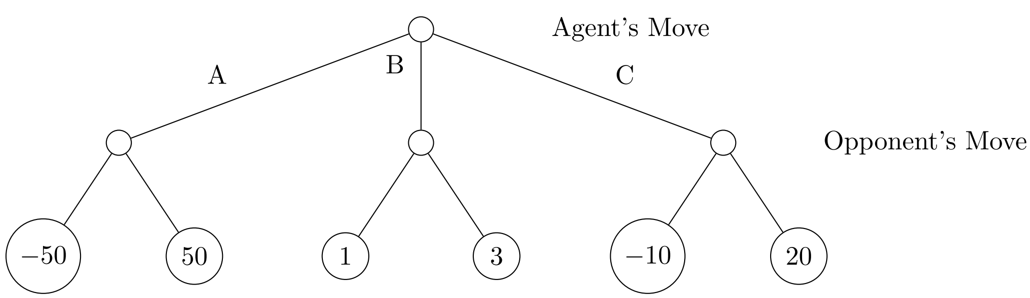
Consider a two-player game in which there are 3 buckets (A, B, C) and each bucket contains two numbers as shown below.

A	**B**	**C**
-50 $\quad$ 50	1 $\quad$ 3	-10 $\quad$ 20

Rules of the game are:

- Player 1(agent) picks the bucket
- Player 2 (opponent) picks a number from the "Selected" Bucket
- Player 1 (agent) utility will be the number picked by player 2

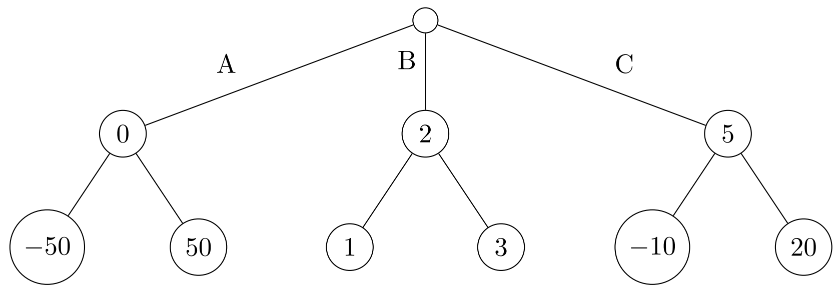
Now, let's construct the **game tree**



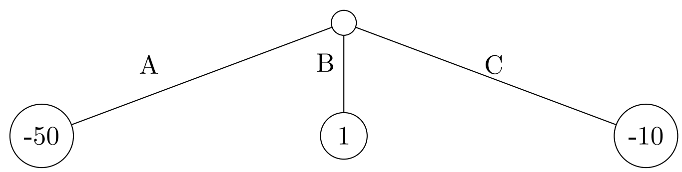
Consider two strategies of the opponent:

1. If the **opponent is stochastic**, i.e., the probability of choosing both left and right numbers in a bucket are the same (i.e., $\frac{1}{2}$).
   - If A is chosen, expected utility $=\frac{-50+50}{2} = 0$.
   - If B is chosen, expected utility $= \frac{1+3}{2} = 2$.
   - If C is chosen, expected utility $= \frac{-10+20}{2} = 5$.

So in this case **agent picks bucket C**, to maximize it's utility.



2. If the opponent is a min player, i.e., the **opponent always chooses the minimum number from the bucket.**
   In this case,the agent **picks Bucket B**, as argmax$(-50,\ 1, -10) = 1$.



Note: Here, utility is given only to the agent. However, in some games, utility can be assigned to both the agent and the opponent. In such scenarios, the opponent not only tries to minimize the agent's score but also to maximize its own utility.

19.2 Two Player Zero Sum Game

Also called **Sequential Move game**. In a sequential move game, players make decisions one after the other, with each player aware of the prior decisions made.

Definitions:

- **Players:** $\{\text{Agent}, \text{Opponent}\}$
- **Starting State:** S_0
- **Actions(s):** Possible actions at state 's'
- **Player(s):** Player who makes the move at state 's'
- **Successor(s, a):** Resulting state if action a is taken at state 's'
- **isEnd(s):** Is state an end state/ Flag to identify terminal state
- **Utility(s):** Agent's utility at the "end state"

Note:
i) In Actions(s), Player(s), Successor(s, a), 's' is an intermediate state
ii) Utility(s) is not defined in other (intermediate) states

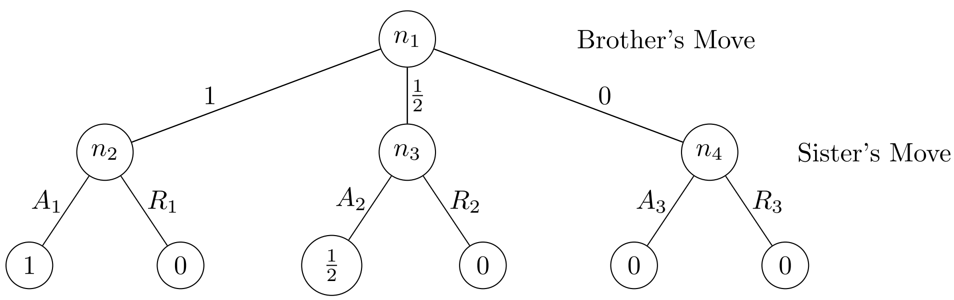
Example: Chess

- **Players:** $\{\text{White}, \text{Black}\}$
- **State:** A board position (position of the chess pieces at a given time)
- **Actions**(s)**:** All legal moves possible by Player(s)
- **Player**(s)**:** Player who makes the move at state 's'
- **Successor**(s, a)**:** Resulting state if action a is taken at state 's'
- **isEnd**(s)**:** Whether 's' is a checkmate or a draw
- Utility function is defined as

$$\textbf{Utility}(s) = \begin{cases} +M & \text{if white wins} \\ -M & \text{if black wins/white loses} \\ 0 & \text{if it is a draw} \end{cases}$$

19.3 Constant Sum Game

Game of a brother and sister sharing two chocolates
Players = {Brother, Sister}



Brother plays first and goes to one of the states n_2, n_3, n_4.
The **state** n_2 means he decides to keep both chocolates.
The **state** n_3 means he decides to keep one and give his sister the other one.
The **state** n_4 means he decides to give his sister both chocolates.

Then his sister chooses A_1 or R_1 if he chose n_2; A_2 or R_2 if he chose n_3; and A_3 or R_3 if he chose n_4.
A_1, A_2, A_3 are **accepting states** - Whatever her brother decided happens.
R_1, R_2, R_3 are **rejecting states -** If she chooses one of these none of them get any chocolate irrespective of the decision made by her brother earlier.

The nodes in the last level signify the utility according to the brother.
The **state** 1 means he gets both chocolates.
The **state** $\frac{1}{2}$ means he gets both the chocolates.
The **state** 0 means his sister gets both chocolates.

$$\textbf{Utility}(s) = \begin{cases} \text{Gets both the chocolates} & \text{if end state is 1} \\ \text{Gets one chocolate} & \text{if end state is 1/2} \\ \text{Gets no chocolates} & \text{if end state is 0} \end{cases}$$

$$\textbf{Actions}(s) = \begin{cases} \{1, \frac{1}{2}, 0\} & \text{when s} = n_1 \\ \{A_1, R_1\} & \text{when s} = n_2 \\ \{A_2, R_2\} & \text{when s} = n_3 \\ \{A_3, R_3\} & \text{when s} = n_4 \end{cases}$$

`Note:` The utility function is defined for the brother

19.4 Strategy of a player

19.4.1 Deterministic Strategies:

In deterministic strategies, **the action of a player at a specific state is fixed and predetermined**. For any given state S, the action chosen by player i is strategically fixed, denoted by $\pi_i(s)$, and it belongs to the set of all possible actions available at that state, provided player(s) = i i.e.,

$$\pi_i(s) \in \text{actions}(s) \quad \text{if} \quad \text{player}(s) = i$$

19.4.2 Randomized Strategies:

In contrast, **randomized strategies incorporate probabilities into the decision-making process**. For a state S, the strategy of the player for choosing an action is probabilistic. The strategy $\pi_i(s)$ assigns a probability to each possible action, meaning that $\pi_i(s)$ belongs to the set of all probability distributions over the set of actions available at state S, provided player(s) = i i.e.,

$$\pi_i(s) \in \Delta(\text{actions}(s)) \quad \text{if} \quad \text{player}(s) = i$$

where $\Delta(A)$ denotes the set of all probability distributions over the set A, which in this context, is the set of actions available at state S.

Consider the above game of brother and sister sharing two chocolates, and at state n_1, if the brother's strategy is stochastic,

$$\pi_B(n_1) = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$$

This implies that for the state n_1, the brother's strategy assigns equal probabilities to each possible action (A, B, C). Specifically, each action (A, B, C) has a 1/3 chance of being chosen.

19.5 Games with partial information

$$u_{agent}(s) = \begin{cases} \text{utility(s)} & \textit{if } \text{isEnd(s)} \\ \sum_{a \in actions(s)} \Pi_{agent}(s)[a] \ \text{u}_{agent}(\text{successor}(s, a)) & \textit{if } \text{player(s)} = \text{agent} \\ \sum_{a \in actions(s)} \Pi_{opponent}(s)[a] \ \text{u}_{agent}(\text{successor}(s, a)) & \textit{if } \text{player(s)} = \text{opponent} \end{cases}$$

`Note:` $\Pi_{agent}(s)[a]$ and $\Pi_{opponent}(s)[a]$ are probabilities that the agent and the opponent pick action 'a' respectively.

→ `If the opponent is stochastic but the player is a utility maximizer:`
Then the player is called a **'Max player'**. Now the utility of the agent changes to

$$u_{agent}(s) = \begin{cases} utility(s) & \textit{if } \text{isEnd(s)} \\ \max_{a \in actions(s)} \ \mathrm{u}_{agent}(\text{successor}(s,a)) & \textit{if } \text{player(s)} = \text{agent} \\ \sum_{a \in actions(s)} \Pi_{opponent}(s)[a] \ \mathrm{u}_{agent}(\text{successor}(s,a)) & \textit{if } \text{player(s)} = \text{opponent} \end{cases}$$

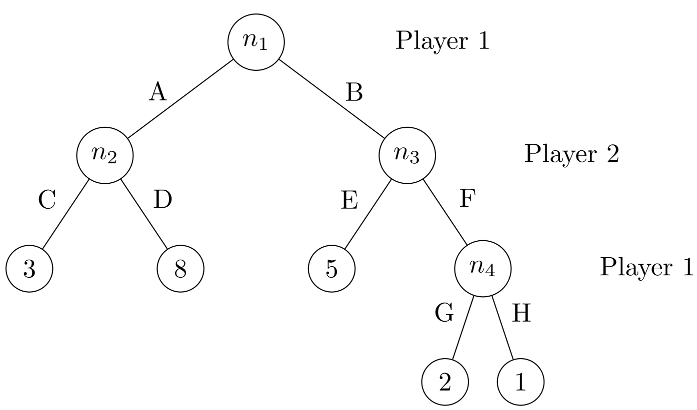
→ `If the opponent is a utility minimizer:`
Then the opponent is called a **'Min player'**. Now the utility of the agent changes to

$$u_{agent}(s) = \begin{cases} utility(s) & \textit{if } \text{isEnd(s)} \\ \max_{a \in actions(s)} \ \mathrm{u}_{agent}(\text{successor}(s,a)) & \textit{if } \text{player(s)} = \text{agent} \\ \min_{a \in actions(s)} \ \mathrm{u}_{agent}(\text{successor}(s,a)) & \textit{if } \text{player(s)} = \text{opponent} \end{cases}$$

Q : Is $\Pi^{\text{max min}}_{\text{agent}}$ the optimal strategy when $\Pi^{\text{stochastic}}_{\text{opponent}}$?
A : No, we don't have an optimal policy, rather we have an equilibrium ($\Pi^{\text{max min}}_{\text{agent}}$, $\Pi^{\text{min}}_{\text{opponent}}$).

Q : Is ($\Pi^{\text{stochastic}}_{\text{agent}}$, $\Pi^{\text{stochastic}}_{\text{opponent}}$) an equilibrium ?
A : No.

19.6 Subgame and Subgame Perfection

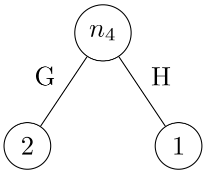


A subgame at 'S' is a restriction of the game at the subtree rooted at 'S' where isEnd(s) is false.

Here, **Player 1** is a utility maximizer (**Max Player**) i.e. plays to maximize his utility.
Hence if player(s) = player 1, the utility of Player 1 changes to,
$u_1(s) = \max_{a \in actions(s)} u_1(\text{successor}(s,a))$

Player 2 is the utility minimiser(**Min Player**) i.e. plays to minimize his utility.
Hence if player(s) = player 2, the utility of Player 1 changes to,
$u_1(s) = \min_{a \in actions(s)} u_1(\text{successor}(s,a))$

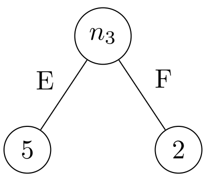
Let us look at the **subgame at node** n_4 :



In this subgame, it's Player 1's turn and he is a max player.
Player 1 (max player) should **pick 'G'** here so as to get '2' (>1, that he would have got by choosing 'H').
We are done solving this subgame.

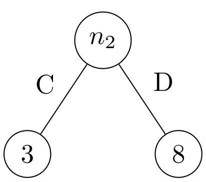
Similarly `using the result of the subgame at` n_4, we can **solve the subgame at** n_3, and this way we move to the upper levels.

Let us take a look at the **subgame at** n_3 :



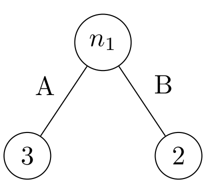
In this subgame, it's Player 2's turn and he is a min player.
Therefore, he should **pick 'F'** so that player 1 gets '2' (from n_4, instead of getting '5' by picking 'E').

Let us look at the **subgame at** n_2 :



In this subgame, it's Player 2's turn and he is a min player.
Therefore, he should **pick 'C'** so that player 1 gets '3' (instead of getting '8' by picking 'D').

Now, let us take a look at the **subgame at** n_1 :



In this subgame, it's Player 1's turn.
Player 1 (max player) should **pick 'A'** here to get '3' (>2, that he would have got by choosing 'B').
Therefore, the **final utility of the Player 1 = 3**

Subgame Perfect Equilibrium is an equilibrium at every subgame/subtree.

The **summary of solving the above game** is:

- Player 1 will choose 'H' in the subgame at n_4
- Player 2 will choose 'F' in the subgame at n_3
- player 2 will choose 'C' in the subgame at n_2
- player 1 will choose 'A' in the subgame at n_1

19.7 Backward Induction

```
function BackInd(s):
    if isEnd(s):
        return u_agent, φ                       // empty set as we don't have any action here

    if player(s) = agent:                      // try to maximize utility
        bestUtility = −∞
        for all a ∈ actions(s) do:
            utilityAtChild, bestAvector ← BackInd(succ(s,a))
            if utilityAtChild > bestUtility:
                bestUtility = utilityAtChild
                bestAvector = append(a, bestAvector)

    if player(s) = opponent:                   // try to minimize utility
        bestUtility = ∞
        for all a ∈ actions(s) do:
            utilityAtChild, bestAvector ← BackInd(succ(s,a))
            if utilityAtChild < bestUtility:
                bestUtility = utilityAtChild
                bestAvector = append(a, bestAvector)

    return bestUtility, action_vector(bestAvector)
```

We can apply Backward induction on small games like Tic-tac-toe.
But can apply it to Chess, Go, Checkers, ...?
We can, but the game tree is huge.

Checkers game tree $\sim 10^{20}$ nodes
Chess game tree $\sim 10^{40}$ nodes
Go game tree $\sim 10^{170}$ nodes
Checkers was solved in 2007 after 18 years of computation and the optimal solution was a Draw.