Artificial Intelligence

BS (CS) _Spring_2025

Lab_10 Manual

Learning Objectives:

1. Adversarial Search (Alpha Beta Pruning)

Lab Manual

Learning Objectives:

- Understand and implement Adversarial Search using the Alpha-Beta Pruning technique.
- Enhance the efficiency of game-playing agents using optimization strategies in game trees.

Adversarial Search with Alpha-Beta Pruning

Adversarial search is used when more than one agent is actively competing in the same search space, such as in two-player games. Unlike traditional single-agent search algorithms that aim to find a sequence of actions to reach a goal, adversarial searches simulate competition, where each agent considers the actions of the other and their impact.

Such environments are called multi-agent environments, and adversarial search is crucial in modeling and solving problems in these domains.

Game Playing in AI

Game playing is a well-known domain for applying adversarial search techniques.

Games have:

- Clear rules
- Defined win/loss conditions
- Legal moves that can be programmatically generated

Search techniques like Breadth-First Search (BFS) or Depth-First Search (DFS) become inefficient due to the high branching factor. Instead, Minimax Search is typically used.

However, Minimax has its own limitations due to the exhaustive nature of tree traversal. This is where Alpha-Beta Pruning enhances performance by reducing unnecessary node evaluations.

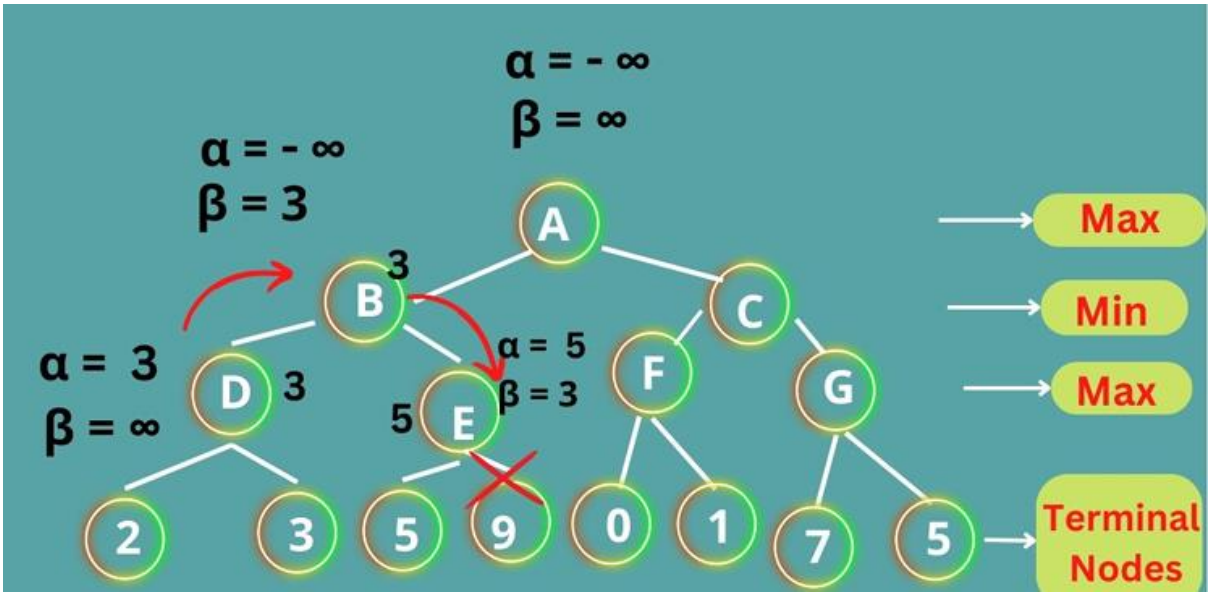
Alpha-Beta Pruning:

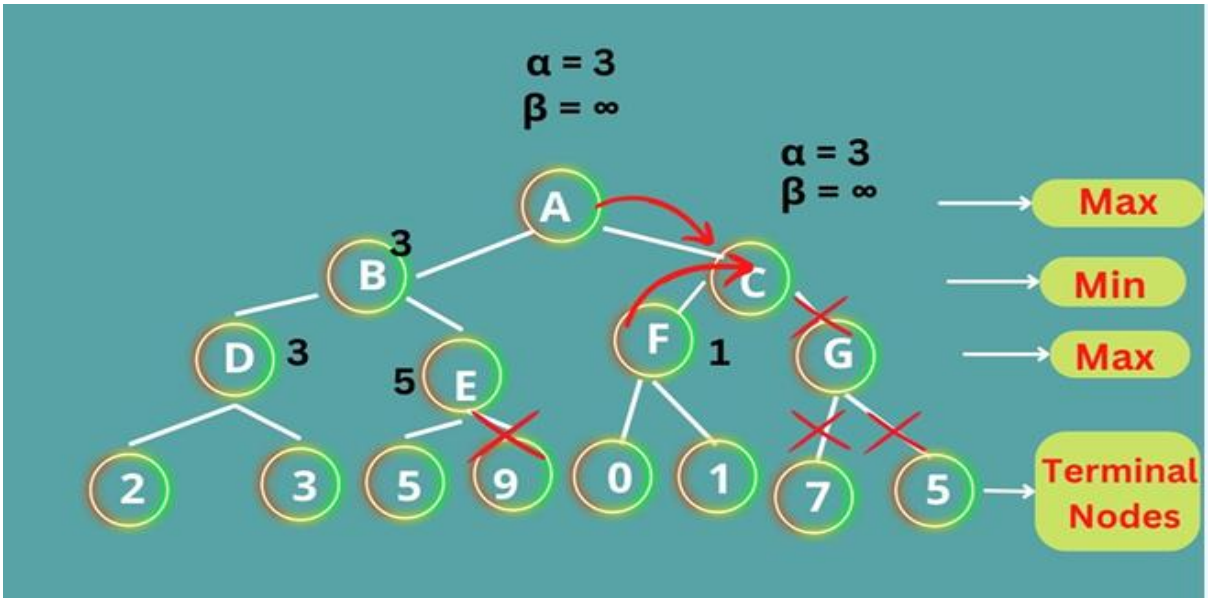
Alpha beta pruning in Artificial Intelligence is a way of improving the minimax algorithm. In the minimax search method, the number of game states that it must investigate grows exponentially with the depth of the tree. We cannot remove the exponent, but we can reduce it by half. As a result, a technique known as pruning allows us to compute the proper minimax choice without inspecting each node of the game tree. Alpha-beta pruning may be done at any level in a tree, and it can occasionally trim the entire sub-tree and the tree leaves. It is named alpha-beta pruning in artificial intelligence because it involves two parameters, alpha, and Beta.

Condition for Alpha Beta Pruning

The two parameters of alpha beta pruning in artificial intelligence are

- **Alpha:** It is the best highest value choice that we have found in the path of the maximizer. The starting value of alpha is -∞.
- **Beta:** It is the best lowest value choice that we have found in the path of the minimizer. The starting value of beta is +∞.
- The condition for Alpha-beta Pruning is α >= β.
- Alpha is updated only when it's MAX's time, and Beta can only be updated when it's MIN's turn.





Pseudocode for Alpha-Beta Pruning:

```
def alphabeta(node, depth, alpha, beta, maximizingPlayer):
    if depth == 0 or TERMINAL_TEST(node):
        return EVALUATION(node)

    if maximizingPlayer:
        value = -INFINITY
        for child in MOVE_GEN(node):
            value = max(value, alphabeta(child, depth-1, alpha, beta, False))
            alpha = max(alpha, value)
            if beta <= alpha:
                break  # Beta cut-off
        return value
    else:
        value = +INFINITY
        for child in MOVE_GEN(node):
            value = min(value, alphabeta(child, depth-1, alpha, beta, True))
            beta = min(beta, value)
            if beta <= alpha:
                break  # Alpha cut-off
        return value
```

Key Components

- **MOVE-GEN (state):** Returns all legal moves for the current state.
- **EVALUATION (state):** Scores a game state from the maximizing player's perspective.
- **TERMINAL-TEST (state):** Checks if the game has reached a win, loss, or draw.
- **Alpha and Beta:** Bounds passed during recursive evaluation to control pruning.

How Alpha-Beta Works (Step-by-Step)

- Start from the current game state.
- Generate all legal moves.
- Recursively evaluate these moves using Minimax.
- During evaluation, keep track of alpha and beta values.
- If a branch cannot improve the result, cut it off.
- Return the move leading to the best guaranteed outcome.