ABHYASMITRA.IN
STUDY NOTES

ARTIFICIAL INTELLIGENCE

Unit III — Adversarial Search, Game Theory, and CSP

Topics Covered (09 Hours)

Adversarial Search & Optimal Decisions in Games (Minimax)
Heuristic Alpha–Beta Tree Search (Theory & Math)
Monte Carlo Tree Search (Selection, Simulation, Playout)
Stochastic Games (Expectiminimax & Chance Nodes)
Partially Observable Games & Belief States
Limitations of Game Search Algorithms (Horizon Effect)
Constraint Satisfaction Problems (CSP) Formulation
Constraint Propagation & Arc Consistency (AC-3 Algorithm)
Backtracking Search for CSPs & Optimization Heuristics

Compiled in a simple, easy-to-understand, book style format
for students of Computer Science and Engineering

TABLE OF CONTENTS

3.1 Optimal Decisions in Games

In many Artificial Intelligence tasks, the agent acts in a competitive environment where other agents are working against it. These environments are called adversarial search problems, commonly known as games. In game theory, we analyze situations where multiple players make decisions, each trying to maximize their own utility. The most common type of game in classical AI is a deterministic, two-player, turn-taking, zero-sum game of perfect information (such as Chess, Checkers, or Tic-Tac-Toe).

Zero-Sum Game
A zero-sum game is a mathematical representation of a conflict where one player's gain is exactly equal to the other player's loss. The total utility of all players sums to zero at the end of the game.

The Minimax Algorithm

To find the optimal sequence of moves, we represent the game as a game tree. The root node represents the current board state, and the branches represent the moves available. We define two players: MAX (who wants to maximize the final score) and MIN (who wants to minimize the final score).

The Minimax Algorithm computes the optimal decision for MAX by recursively calculating the minimax value of each child node:

- At a leaf node (terminal state), the minimax value is the utility of that state.
- At a MAX node, the minimax value is the maximum of the values of its children.
- At a MIN node, the minimax value is the minimum of the values of its children.

Properties of the Minimax Algorithm

Criterion	Status	Detailed Explanation
Completeness	Yes	If the game tree is finite, the algorithm will always find a solution path.
Time Complexity	O(b^d)	It explores the entire game tree. b is the branching factor and d is the maximum depth of the tree.
Space Complexity	O(b*d)	It performs a depth-first search, needing only to store the current path of actions.
Optimality	Yes	It is optimal against an optimal opponent. If the opponent plays suboptimally, MAX will achieve an even higher score.

3.2 Heuristic Alpha–Beta Tree Search

The time complexity of Minimax is O(b^d), which is completely impractical for complex games like Chess (where b is around 35 and d can be over 80). To solve this, we use a technique called Alpha-Beta Pruning. This process cuts off branches of the search tree that cannot possibly affect the final decision, allowing us to search much deeper.

Alpha-Beta Pruning Parameters

The algorithm maintains two values throughout the recursive search:

α (alpha): The value of the best (highest-value) choice found so far at any choice point along the path for MAX.

β (beta): The value of the best (lowest-value) choice found so far at any choice point along the path for MIN.

The Pruning Condition

As the search proceeds, if the algorithm finds a node where alpha is greater than or equal to beta, it stops evaluating the remaining children of that node. This is because the opponent (MIN) can already force a worse outcome elsewhere, so the current path is irrelevant.

Prune when: **α ≥ β**

Heuristic Evaluation Functions

In real-world games, we cannot search all the way to terminal states. Instead, we use a Cutoff Test to stop the search at a certain depth and apply an Evaluation Function to estimate the utility of the state. A good evaluation function estimates the probability of winning from that position (e.g., in Chess, assigning weights to pieces: pawn = 1, knight = 3, queen = 9).

3.3 Monte Carlo Tree Search

In games with massive branching factors like Go (where b is around 250), alpha-beta pruning is still too slow, and it is extremely difficult to write a good heuristic evaluation function. Monte Carlo Tree Search (MCTS) solves this by using random simulations to evaluate positions, rather than static heuristic calculations.

The Four Phases of MCTS

- **1. Selection:** Starting at the root node, the algorithm navigates down the tree using a selection policy (like Upper Confidence Bound, UCB1) that balances exploitation (picking known good nodes) and exploration (visiting rarely analyzed nodes) until it reaches a leaf node.
- **2. Expansion:** Unless the leaf node represents a terminal state, the algorithm creates one or more new child nodes to expand the search tree.
- **3. Simulation (Rollout):** From the newly created node, the algorithm runs a simulated game (playout) by making random or semi-random moves until a terminal game state (win, loss, or draw) is reached.
- **4. Backpropagation:** The result of the simulation (win/loss) is sent back up the path to the root, updating the visit count and win statistic of all nodes along the way.

> **NOTE**
> MCTS does not need a hand-crafted evaluation function. The value of a node is determined dynamically by the win rate of the random playouts. MCTS is the core algorithm behind Google DeepMind's historic AlphaGo program.

3.4 Stochastic Games

In the real world, many games involve an element of chance, such as rolling dice, shuffling cards, or weather events. These are called stochastic games. An example is Backgammon, where players roll dice before choosing their moves.

The Expectiminimax Algorithm

To handle stochastic environments, we modify the minimax tree to include chance nodes between the MAX and MIN layers. The branches leaving a chance node represent the possible random outcomes (e.g., the possible dice rolls), each labeled with its probability of occurrence.

- At a MAX node, we calculate the maximum utility of the child nodes.
- At a MIN node, we calculate the minimum utility of the child nodes.
- At a chance node, we calculate the expected value (average utility) of all child nodes, weighting each child's value by its probability.

> **NOTE**
> Stochastic search is computationally demanding. Because chance nodes add many branches, the search tree grows extremely fast, limiting the depth that expectiminimax can reach in real-time play.

3.5 Partially Observable Games

In partially observable games, players do not have full access to the state of the board. Examples include card games like Poker (where you cannot see the opponent's cards) or Battleship (where you cannot see the position of enemy ships).

Because the actual state is unknown, an agent must maintain a belief state—a probability distribution over all possible actual states. When planning a move, a partially observable agent evaluates how its actions will affect its belief state. For example, in Poker, a bet is not just about the cards in hand; it is a signal that alters the opponent's belief state about your hand.

3.6 Limitations of Game Search Algorithms

While game search algorithms have achieved super-human performance in many board games, they suffer from several fundamental limitations:

- **The Horizon Effect:** This occurs when a threat from the opponent is inevitable but lies just beyond the search limit (the horizon). The algorithm will select moves that temporarily postpone the threat, often making its physical position worse, rather than dealing with the threat directly.
- **Combinatorial Explosion:** Even with alpha-beta pruning, the number of nodes to search grows exponentially with depth, making deep lookaheads impossible in highly complex games.
- **Heuristic Inaccuracy:** If the evaluation function is poorly written or contains a bias, A* or minimax will optimize for the wrong criteria, leading to strategic failures.

3.7 Constraint Satisfaction Problems (CSP)

In standard search, states are treated as 'black boxes'—they can be accessed only by the transition model and goal test. In contrast, Constraint Satisfaction Problems (CSPs) represent states in a structured format, using variables, domains, and constraints.

CSP Component Definition

A Constraint Satisfaction Problem consists of three components:

1. A set of Variables, X = {X1, X2, ..., Xn}

2. A set of Domains, D = {D1, D2, ..., Dn}, specifying the allowable values for each variable.

3. A set of Constraints, C, that specify the allowed combinations of values.

Example: Map Coloring

Consider the task of coloring a map of Australia using three colors (Red, Green, Blue) such that no two adjacent regions share the same color.

- **Variables:** The regions: X = {WA, NT, Q, NSW, V, SA, T}.
- **Domains:** The colors for each region: Di = {red, green, blue}.
- **Constraints:** Adjacent regions must have different colors. For example: WA != NT.
- **Solution:** An assignment of values to variables that is complete (all variables have values) and consistent (all constraints are satisfied).

3.8 Constraint Propagation: Inference in CSPs

Instead of immediately searching, a CSP agent can solve problems by performing inference, called Constraint Propagation. This process uses constraints to reduce the number of legal values in a variable's domain, simplifying the problem before or during the search.

Types of Consistency

- **Node Consistency:** A variable is node-consistent if all the values in its domain satisfy the variable's unary constraints (constraints involving only that variable).
- **Arc Consistency (AC-3 Algorithm):** A variable Xi is arc-consistent with respect to another variable Xj if, for every value in the domain Di, there is some value in Dj that satisfies the binary constraint between them. The AC-3 algorithm repeatedly checks all arcs, removing inconsistent values from domains until no further reductions can be made.
- **Path Consistency:** Looks at triples of variables. A pair of variables {Xi, Xj} is path-consistent with respect to a third variable Xk if, for every relation between Xi and Xj, there is a path through Xk that is also compatible.

3.9 Backtracking Search for CSPs

Backtracking Search is the basic uninformed algorithm for solving CSPs. It is a depth-first search that chooses values for one variable at a time, backtracking when a variable has no legal values left to assign.

Heuristics to Improve Backtracking

To make backtracking highly efficient, we use several general-purpose heuristics:

- **Minimum Remaining Values (MRV):** Also called the 'most constrained variable' heuristic. It chooses the variable with the fewest 'legal' values remaining in its domain. This minimizes the branching factor.
- **Degree Heuristic:** Used to break ties. It chooses the variable that has the most constraints on other unassigned variables, helping to reduce future choices quickly.
- **Least Constraining Value (LCV):** Once a variable is chosen, it selects the value that rules out the fewest choices for the neighboring unassigned variables, leaving maximum flexibility.
- **Forward Checking:** Whenever a variable X is assigned a value, the algorithm looks at all unassigned variables connected to X and deletes any value from their domains that is inconsistent with the assignment, catching failures early.

Unit Summary

This unit discussed adversarial search, game theory, and constraint satisfaction problems. The key takeaways are summarized below.

- Adversarial search algorithms analyze competitive environments (games) where players attempt to maximize their own utility.
- Minimax computes the optimal move for MAX by recursively calculating maximum and minimum values down a game tree.
- Alpha-Beta Pruning speeds up Minimax by discarding branches where alpha >= beta, allowing much deeper search limits.
- Monte Carlo Tree Search (MCTS) bypasses heuristic functions by running random playouts through selection, expansion, simulation, and backpropagation.
- Stochastic games incorporate chance elements using expectiminimax, which averages child values at chance nodes based on probabilities.
- Partially observable games require tracking belief states over unknown configurations (e.g., opponent cards).
- Constraint Satisfaction Problems (CSPs) structure states into variables, domains, and constraints, rather than black boxes.
- Constraint Propagation (like the AC-3 arc-consistency algorithm) simplifies CSPs by removing illegal values from variable domains.
- Backtracking is a DFS-style search for CSPs that is optimized using MRV, Degree, LCV heuristics, and Forward Checking.