

AI ENDSEM UNIT – 3 PYQ

➤ **MAY / JUN 2022**

Q1)

a) Explain Alpha - Beta Tree search and cutoff procedure in detail with example.

Alpha-Beta Tree Search

Alpha-Beta Pruning is an optimization technique for the Minimax algorithm used in adversarial games (like chess or tic-tac-toe).

It reduces the number of nodes that are evaluated in the search tree by eliminating branches that won't affect the final decision.

Key Terms:

- **Alpha (α):** The best value that the maximizer can guarantee so far.
- **Beta (β):** The best value that the minimizer can guarantee so far.
- **Pruning:** Cutting off branches in the tree that won't be selected in the optimal move.

Working of Alpha-Beta Pruning:

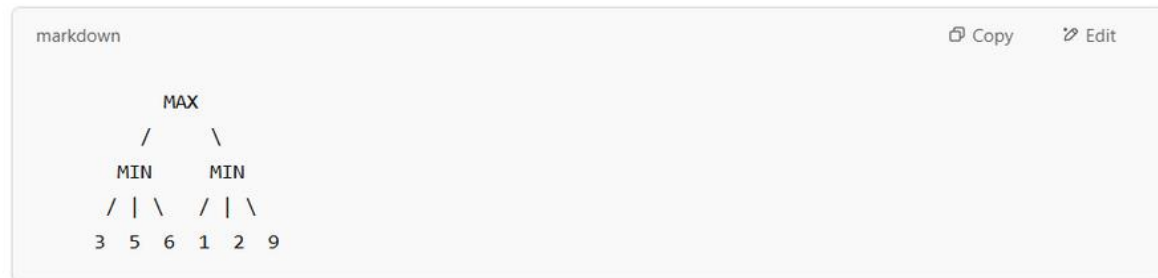
1. Traverse the game tree in depth-first manner.
2. At MAX nodes, update α and prune if $\alpha \geq \beta$.
3. At MIN nodes, update β and prune if $\beta \leq \alpha$.
4. It ensures the best decision is made without checking all branches.

Cutoff Procedure:

- Cutoff occurs when a branch is guaranteed to be worse than a previously examined branch.
- It helps in reducing the search time by not exploring unnecessary path.

Example:

Consider a minimax tree:



Let's apply **Alpha-Beta Pruning**:

1. Start from left MIN node:

- Explore 3 → MIN = 3 → $\beta = 3$
- Explore 5 → MIN = $\min(3,5) = 3$ → $\beta = 3$
- Explore 6 → MIN = $\min(3,6) = 3$

→ Left MIN subtree value = 3

→ MAX $\alpha = 3$

2. Go to right MIN node:

- First child = 1 → $\beta = 1$
→ Now $\alpha = 3$ and $\beta = 1$ → Since $\beta \leq \alpha$, prune remaining nodes (2, 9).

→ **Pruned** due to cutoff.

Result:

- Final decision: MAX chooses the left branch with value = 3.
- Nodes 2 and 9 are not evaluated, saving computation.

Advantages:

- Reduces **time complexity** from $O(b^d)$ to $O(b^{(d/2)})$ in best case.
- Makes game-playing agents faster and more efficient.

b) What are the issues that need to be addressed for solving csp efficiently? Explain the solutions to them. [9]

Issues to Address for Solving CSPs Efficiently:

1. **Variable Ordering**
Choosing the order in which variables are assigned has a huge impact on efficiency.
2. **Value Ordering**
Deciding the order of values to assign from a domain can reduce backtracking.
3. **Constraint Propagation**
Ensuring consistency between variables **early** helps avoid future conflicts.
4. **Backtracking Problems**
Naïve backtracking can be very slow, leading to combinatorial explosion.
5. **Early Detection of Failure**
Without inference, failures are detected late, wasting time.
6. **Redundant Search Paths**
The algorithm may explore similar paths that lead to the same result.

Solutions to Improve CSP Solving:

1. Variable Ordering Heuristics:

- **Minimum Remaining Values (MRV):** Choose the variable with the fewest legal values left (most constrained).
→ Reduces chances of failure.
- **Degree Heuristic:** If tie in MRV, choose variable involved in most constraints with unassigned variables.

2. Value Ordering Heuristics:

- **Least Constraining Value (LCV):** Choose the value that rules out the fewest choices for neighboring variables.

3. Constraint Propagation:

- Use techniques like:
 - **Forward Checking:** After assigning a value, eliminate inconsistent values from neighbors' domains.
 - **Arc Consistency (AC-3 Algorithm):** Ensures that for every value of variable A, there exists a consistent value in variable B.

4. Improved Backtracking:

- **Backjumping:** Jumps back more than one level to the variable causing the failure.

- **Maintaining Arc Consistency (MAC):** Combine backtracking with constraint propagation for faster solving.

5. Structure-Based Methods:

- Decompose the CSP into **independent subproblems** (e.g., tree-structured CSPs).
- Use **tree decomposition** techniques to simplify constraint graphs.

Efficient CSP solving requires **smart variable selection, value choices, constraint propagation, and improved backtracking.**

These techniques help in **reducing the search space**, avoiding failure early, and solving problems faster.

Q2)

a) Explain in detail the concepts of back tracking and constraint propagation and solve the N-queen problem using these algorithms. [9]

Backtracking

Backtracking is a **systematic trial-and-error** algorithm that incrementally builds solutions and abandons a path ("backtracks") as soon as it determines it cannot lead to a valid solution.

- It is a depth-first search strategy.
- Used for problems with constraints, like Sudoku, N-Queens, etc.

Constraint Propagation

Constraint propagation is a technique used to reduce the search space by enforcing constraints early.

- It narrows down the possible values (domain) of variables.
- Often used with arc consistency (AC-3) and forward checking.
- Helps in early detection of failure before backtracking is needed.

N-Queen Problem Using Backtracking & Constraint Propagation

Problem: Place N queens on an $N \times N$ chessboard such that **no two queens** attack each other (i.e., no same row, column, or diagonal).

Backtracking Algorithm for N-Queens:

1. Start from the first row.
2. Try placing a queen in each column one by one.
3. For each placement, check if it's **safe** (not attacked).
4. If safe, move to the next row (recursive call).
5. If no valid column, backtrack to previous row and change queen's position.

Constraint Propagation with Forward Checking:

- When placing a queen in a column, **eliminate** conflicting columns for the next row.
- Forward checking reduces the domain of future variables (columns).
- If domain of any variable becomes empty, backtrack immediately.

Example: 4-Queen Problem (brief)

Board: 4x4, rows = variables, columns = values.

1. Place queen at (0,1) → forward check: row 1 can't have 1, 0, 2
2. Place at (1,3) → forward check: row 2 can't have 1, 2, 3
3. Place at (2,0) → forward check: row 3 can't have 0, 1
4. Place at (3,2) → SUCCESS!

Conclusion:

- Backtracking explores all possible placements and retreats when stuck.
- Constraint Propagation helps in pruning invalid choices early.
- Combining both improves efficiency, especially for large N.

b) Write a short note on Monte Carlo Tree search and list its limitations. [5]

Monte Carlo Tree Search (MCTS):

Monte Carlo Tree Search is a heuristic search algorithm used in decision-making problems like games (e.g., Go, Chess, General Game Playing). It builds a search tree incrementally and uses random sampling (Monte Carlo simulations) to evaluate the most promising moves.

Main Steps of MCTS:

1. **Selection:** Traverse the tree from root to a leaf node using a policy (like UCT - Upper Confidence Bound).
2. **Expansion:** If the leaf is not terminal, expand it by adding a new child.
3. **Simulation:** Perform a random (or semi-random) simulation from the new node to the end of the game.
4. **Backpropagation:** Update statistics (like win rate) of all nodes in the path from the leaf to the root.

Limitations of MCTS:

1. **High Computation Time:** Simulations can be time-consuming, especially in complex games.
2. **Poor in Deterministic Domains:** Random playouts may not be informative when exact strategies are needed.
3. **Ineffective with Deep Trees:** For very deep game trees, MCTS might not explore enough depth due to limited simulations.
4. **Performance Depends on Simulation Quality:** Poor simulation policies can lead to bad decisions.
5. **Hard to Use in Partially Observable or Real-Time Games:** It requires a clear model of the game environment.

c) Apply constraint satisfaction method to solve following Problem SEND + MORE = MONEY. (TWO + TWO = FOUR, CROSS+ ROADS= DANGER) [4]

Cryptarithmic Problem:

We need to assign digits (0–9) to letters such that:

markdown

Copy

Edit

```
SEND
+ MORE
-----
MONEY
```

Variables:

S, E, N, D, M, O, R, Y — each letter is a variable.

Domains:

Digits 0 to 9.

But:

- No digit repeats.
- $M \neq 0$ and $S \neq 0$ (no leading zeros).



Constraints:

1. AllDifferent Constraint:

All letters must map to different digits.

2. Sum Constraint (Numerical correctness):

The equation must satisfy:

mathematica

Copy

Edit

```
(1000×S + 100×E + 10×N + D) +
(1000×M + 100×O + 10×R + E) =
(10000×M + 1000×O + 100×N + 10×E + Y)
```

3. Carry Rules:

Each digit column must follow carry-forward arithmetic rules.

Solving via CSP:

- Use **Backtracking Search** to test all digit combinations.
- Apply **Constraint Propagation**:
 - Enforce **AllDifferent** early.
 - Prune inconsistent partial assignments using arithmetic checks.

Solution (One Valid Assignment):

yaml

Copy

Edit

```
9567
+ 1085
= 10652
```

Assignment:

- S = 9
- E = 5
- N = 6
- D = 7
- M = 1
- O = 0
- R = 8
- Y = 2

➤ NOV / DEC 2022

Q1)

- a) Explain Min Max and Alpha Beta pruning algorithm for adversarial search with example.[9]

Adversarial Search:

Used in two-player games (e.g., Chess, Tic-Tac-Toe) where players take alternate moves and try to maximize their win while minimizing the opponent's win.

1. Minimax Algorithm:

- It is a recursive algorithm to find the optimal move for a player assuming the opponent also plays optimally.
- Two players:

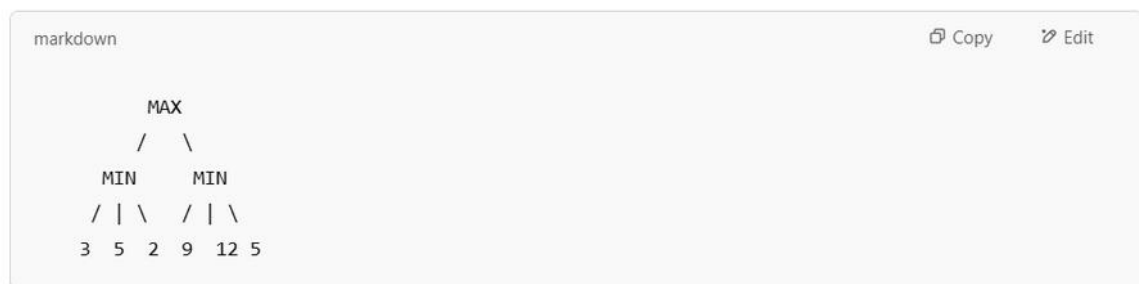
- **MAX:** Tries to maximize the score.
- **MIN:** Tries to minimize the score.

How it works:

- Build the game tree with possible moves.
- At leaf nodes, evaluate the utility values (e.g., win = +1, draw = 0, loss = -1).
- Propagate these values back up the tree:
 - MAX picks the maximum value.
 - MIN picks the minimum value.

Example:

Consider a tree with utility values at leaf nodes:



- Left MIN node $\rightarrow \min(3,5,2) = 2$
- Right MIN node $\rightarrow \min(9,12,5) = 5$
- MAX $\rightarrow \max(2,5) = 5$

So, the best move for MAX is the one that leads to value 5.

2. Alpha-Beta Pruning:

- An optimization of Minimax that **prunes** branches that **cannot affect the final decision**.
- Saves time by avoiding unnecessary evaluations.

Definitions:

- **Alpha (α):** Best value that MAX can guarantee.
- **Beta (β):** Best value that MIN can guarantee.

Pruning Rule:

If at any point $\beta \leq \alpha$, further exploration of that branch is **pruned**.

Example:

Suppose a left branch gives MAX a value of 5. While evaluating the next MIN branch, if a move leads to a value less than 5, MIN will never allow it, so the rest of that subtree is pruned.

Advantages of Alpha-Beta Pruning:

- Same result as Minimax but with **fewer evaluations**.
- In a **balanced tree**, it reduces time complexity from $O(b^d)$ to $O(b^{(d/2)})$, where:
 - b = branching factor
 - d = depth of the tree

Conclusion:

- **Minimax** ensures optimal decision-making in adversarial games.
- **Alpha-Beta Pruning** makes Minimax efficient by skipping unnecessary nodes.

b) Define and explain Constraints satisfaction problem. [9]

A **Constraint Satisfaction Problem (CSP)** is a mathematical problem defined by:

- A set of variables,
- Each having a domain of possible values,
- And a set of constraints that specify allowable combinations of values.

The goal is to assign values to all variables such that all constraints are satisfied.

Components of a CSP:

1. **Variables:**
Denoted by $X=\{X_1, X_2, \dots, X_n\}$
2. **Domains:**
Each variable X_i has a domain D_i , i.e., possible values it can take.
3. **Constraints:**
Conditions or rules restricting the values that variables can take together. These can be:
 - **Unary** (involving 1 variable),
 - **Binary** (2 variables),
 - **n-ary** (more than 2 variables).

Example – Map Coloring Problem:

- **Variables:** {WA, NT, SA, Q, NSW, V, T}
- **Domains:** {Red, Green, Blue}
- **Constraints:** Neighboring states must have different colors, e.g., WA \neq NT, SA \neq Q, etc.

Types of Constraints:

- **Hard constraints:** Must be satisfied (e.g., A \neq B).
- **Soft constraints:** Preferable but not mandatory (used in optimization problems).

Solving Techniques:

1. **Backtracking Search:**

- Try assigning values one by one.
- If a constraint is violated, backtrack and try a different value.

2. **Constraint Propagation:**

- **Forward Checking:** After assigning a value to a variable, remove inconsistent values from neighbouring domains.
- **Arc Consistency (AC-3):** Ensure that for every value of one variable, there is a consistent value in the neighbouring variable.

3. **Heuristics (to improve efficiency):**

- **Minimum Remaining Values (MRV):** Choose the variable with the fewest legal values.
 - **Degree Heuristic:** Choose the variable involved in the most constraints.
 - **Least Constraining Value:** Choose the value that rules out the fewest options for neighboring variables.
-

Applications of CSP:

- Scheduling (e.g., class timetabling)
- Puzzles (e.g., Sudoku, N-Queens)
- Resource allocation
- Map coloring
- Cryptarithmic problems

Q2) a) Explain with example graph coloring problem. [9]

The Graph Coloring Problem is a type of Constraint Satisfaction Problem (CSP) where:

- Each node (vertex) in a graph must be assigned a color.
- No two adjacent nodes (connected by an edge) should have the same color.
- The goal is to color the graph using the minimum number of colors.

Formal CSP Formulation:

- **Variables:** Each vertex in the graph is a variable (e.g., A, B, C, ...).
- **Domains:** The set of colors available (e.g., {Red, Green, Blue}).
- **Constraints:** Adjacent vertices must have different colors.

Map Coloring Example – Indian States

Let's consider 5 neighboring Indian states:

- Rajasthan (RJ)
- Haryana (HR)
- Uttar Pradesh (UP)
- Madhya Pradesh (MP)
- Gujarat (GJ)

Neighbor Relationships (Edges):

State	Neighbors
RJ	HR, UP, MP, GJ
HR	RJ, UP
UP	HR, RJ, MP
MP	RJ, UP, GJ
GJ	RJ, MP

CSP Details:

- **Variables:** {RJ, HR, UP, MP, GJ}
- **Domains:** {Red, Green, Blue} (3 colors)
- **Constraints:** No two neighboring states can have the same color

One Valid Color Assignment:

State	Color
RJ	Red
HR	Green
UP	Blue
MP	Green
GJ	Blue

All neighboring states have different colors — so all constraints are satisfied.

Solving Techniques:

1. **Backtracking Search:** Try assigning colors one by one and backtrack if conflict occurs.

2. **Forward Checking:** Eliminate incompatible colors from neighbors' domains after each assignment.
 3. **Heuristics:**
 - **MRV (Minimum Remaining Values):** Choose the most constrained region first.
 - **Degree Heuristic:** Pick the region with the most neighbors.
 - **Least Constraining Value:** Try the color that leaves most options for others.
-

Applications:

- Map coloring
- Register allocation in compilers
- Scheduling problems (e.g., exams, courses)
- Frequency assignment in networks

b) How AI technique is used to solve tic-tac-toe problem. [9]

Tic-Tac-Toe Game Overview:

- A **two-player, turn-based** game played on a 3×3 grid.
 - Players take turns marking **X** or **O**.
 - The goal is to get **three marks in a row**, column, or diagonal.
 - The game ends in a **win, loss, or draw**.
-

AI Techniques Used to Solve Tic-Tac-Toe:

1. State-Space Representation:

- Each board configuration is a **state**.
- The **initial state** is an empty board.
- Possible **actions**: placing X or O in an empty cell.
- A **transition** is made to a new state after a move.

2. Minimax Algorithm (Adversarial Search):

- Used because the game is **deterministic** and **zero-sum**.
- AI assumes the opponent plays optimally.

Minimax Working:

- **MAX (AI)** tries to **maximize its chance of winning**.
- **MIN (opponent)** tries to **minimize AI's chance**.
- The AI evaluates all possible moves up to terminal states (win/loss/draw), assigns utility values, and **selects the best move**.

Utility Values:

- Win → +1
 - Draw → 0
 - Loss → -1
-

3. Alpha-Beta Pruning (Optimization):

- Used with Minimax to **skip unnecessary branches**.
 - Reduces computation time by **eliminating moves** that won't affect the final decision.
-

4. Heuristic Evaluation (Optional):

- In large games, heuristics are used to **estimate the value** of non-terminal states.
 - For Tic-Tac-Toe, full search is possible due to small state space, so heuristics are usually not required.
-

5. Implementation Steps for AI:

1. Represent board and legal moves.
2. Use **Minimax** to simulate all possible outcomes.
3. Apply **Alpha-Beta Pruning** to speed up decisions.
4. Return the move with the **best utility value**.

Example:

If AI is playing as X and current board is:

```
markdown
x | o |
-----
  | x |
-----
o |  |
```

AI evaluates all possible moves, applies Minimax, and chooses the one leading to a win or blocking opponent's win.

➤ MAY / JUN 2023

Q1)

- a) List All problem solving strategies. What is backtracking, explain with n queen problem, with Branch and bound or Backtracking. [8]

Problem-Solving Strategies in AI:

1. Brute Force Search
2. Heuristic Search (e.g., A*)
3. Greedy Search
4. Backtracking
5. Branch and Bound
6. Constraint Satisfaction
7. Adversarial Search (e.g., Minimax)
8. Hill Climbing and Local Search

What is Backtracking?

Backtracking is a **depth-first search strategy** used for solving constraint satisfaction problems. It **recursively tries** possible solutions and **backtracks** when a constraint is violated.

N-Queen Problem (Using Backtracking):**Problem:**

Place **N queens** on an **N×N chessboard** such that no two queens attack each other (i.e., not in the same row, column, or diagonal).

Algorithm:

1. Start with the **first row**, and try placing a queen in each column.
2. Move to the **next row**, and repeat by checking for a **safe position**.
3. If a conflict occurs (no valid position), **backtrack** to the previous row and try the next column.
4. Continue until all queens are placed or all possibilities are exhausted.

Example for N = 4:

Step-by-step, the backtracking algorithm tries positions:

```
less                                                                    Copy Edit
[Q . . .]      // Row 1, Col 1
[. . Q .]      // Row 2, Col 3
[. Q . .]      // Row 3, Col 2 → conflict (diagonal)
[Backtrack...] // Try next column...
```

Eventually, the solution is:

```
css                                                                    Copy Edit
. Q . .
. . . Q
Q . . .
. . Q .
```

Backtracking vs Branch and Bound:

- **Backtracking:** Tries all possibilities, eliminates those that violate constraints.
- **Branch and Bound:** Also explores nodes like backtracking, but uses **cost bounds** to prune the search space.

In N-Queen, **Backtracking** is more commonly used as it's not an optimization problem but a feasibility one.

Backtracking is an efficient problem-solving strategy for combinatorial problems like **N-Queen**, where it systematically explores possible configurations and eliminates invalid ones via recursion and pruning

b) Explain Monte Carlo Tree Search with all steps and Demonstrate with one Example. [9]

Monte Carlo Tree Search (MCTS):

MCTS is a heuristic search algorithm used for decision-making in games and planning problems. It is especially useful in large game trees like Go, Chess, or general AI planning where traditional search becomes infeasible.

It builds the search tree **asymmetrically**, focusing on **promising moves** by running simulations (random playouts).

Steps of MCTS:

1. **Selection:**
Start at the root and select child nodes **based on a selection policy** like UCT (Upper Confidence Bound). Move down until you reach a node that is **not fully expanded**.
2. **Expansion:**
Add a new child node to the tree by exploring one or more **untried moves** from the selected node.
3. **Simulation (Rollout):**
From the newly added node, **simulate a game till the end** using random or semi-random moves.
4. **Backpropagation:**
Propagate the result of the simulation (win/loss/draw) back through the nodes visited in this iteration, updating their statistics.

These steps are repeated **thousands or millions of times**, and the move with the **highest visit count or win rate** is selected.

Example: Tic-Tac-Toe (Simplified MCTS Illustration)

Suppose it's **X's turn** in the following board:

```

markdown
Copy Edit

x | o |
-----
  | x |
-----
o |   |

```

Step 1: Selection

From root node (current board), MCTS selects a promising move, e.g., placing X at (1, 0).

Step 2: Expansion

Add a child node where X is placed at (1, 0).

Step 3: Simulation

Play randomly from that state: O \rightarrow (2,1), X \rightarrow (0,2), etc., until the game ends in win/loss/draw.

Step 4: Backpropagation

If X wins in the simulation, increase win count of nodes visited. Otherwise, update as per result.

After many such iterations, the move with the **best win rate** (e.g., placing X at (0,2)) will be selected.

Advantages of MCTS:

- Works well in **large search spaces**.
- Doesn't need a complete evaluation function.
- Focuses search on **promising branches**.

Applications:

- Games like Go, Chess, Tic-Tac-Toe.
- AI agents in planning and robotics.
- AlphaGo by DeepMind used **enhanced MCTS + Deep Learning**.

Q2)

a)

i) Explain limitations of game search algorithm, Differentiate between stochastic and partial games

AND.

ii) Explain How use of alpha and beta cut-offs will improve performance of mini max algorithm? [9]

i)

Limitations of Game Search Algorithms:

- 1. **High Computational Complexity:**
 - o Game trees grow **exponentially** with depth and branching factor, making full search infeasible for complex games like chess or Go.
- 2. **Imperfect Information:**
 - o Most algorithms assume **perfect knowledge** of the game state, but many real-world games have hidden information.
- 3. **Stochasticity:**
 - o Random elements (like dice rolls) introduce uncertainty, complicating deterministic search methods.
- 4. **Memory Constraints:**
 - o Storing large game trees is impractical, limiting the depth of search.
- 5. **Evaluation Function Dependence:**
 - o Heuristic evaluation functions can be inaccurate, leading to poor decisions.
- 6. **Time Constraints:**
 - o Limited time forces shallow searches, reducing accuracy.

Aspect	Stochastic Games	Partially Observable Games
Definition	Games with random events affecting outcomes (e.g., dice rolls).	Games where players have incomplete knowledge of the game state.
State Information	Full state is known; randomness affects next state.	State is not fully known due to hidden information.
Example	Backgammon (dice rolls)	Poker (opponent’s cards hidden)
Search Complexity	Requires handling probability in decisions	Requires reasoning over possible hidden states
Algorithms Used	Expectiminimax, Monte Carlo Tree Search	Belief-state search, Partially Observable Markov Decision Processes (POMDPs)

ii) How Alpha-Beta Cut-offs Improve Minimax Performance

- **Alpha-Beta pruning** is an optimization technique that **prunes branches** in the minimax search tree that cannot possibly affect the final decision.
- **Alpha (α)**: Best value found so far for the **maximizing player**.
- **Beta (β)**: Best value found so far for the **minimizing player**.
- **Process:**
 - While searching, if the current node's value is worse than α or β (depending on maximizing/minimizing), the branch is **cut off** (not explored further).
- **Benefits:**

1.Reduces nodes evaluated drastically — from $O(b^d)$ to about $O(b^{(d/2)})$ in the best case

(b = branching factor, d = depth).

1. Enables **deeper searches in the same time**.
 2. Maintains the **same optimal move** as basic minimax (no loss in accuracy).
-

Summary:

- Alpha-Beta pruning **improves efficiency without affecting outcome**, making minimax practical for larger games.
- Limitations of game search algorithms include high complexity, imperfect information, and reliance on heuristics.
- Stochastic games include random chance, while partially observable games involve hidden information.

b) Define is Constraint satisfaction problem, State the types of consistencies Solve the following Crypt Arithmetic Problem. SEND +MORE MONEY [8]

1. Definition of Constraint Satisfaction Problem (CSP):

A **Constraint Satisfaction Problem (CSP)** is a problem defined by:

- **Variables:** A set of variables $X=\{X_1,X_2,...,X_n\}$
- **Domains:** Each variable X_i has a domain D_i of possible values.
- **Constraints:** A set of rules that specify allowable combinations of values for subsets of variables.

The goal is to **assign values** to variables such that **all constraints are satisfied**.**2. Types of Consistencies in CSP:**

a) Node Consistency

Each variable satisfies its **unary constraints** (e.g., variable $X \neq 0$).

b) Arc Consistency (AC)

For every value of variable X, there exists a consistent value in variable Y (binary constraints).

c) Path Consistency

Ensures consistency among **three variables**: if X and Y are consistent, and Y and Z are consistent, then X and Z should also be consistent.

3. Solve the Crypt Arithmetic Problem

SEND + MORE = MONEY

Each letter represents a unique digit (0–9), and:

- $S \neq 0, M \neq 0$ (leading digits cannot be 0)
- All letters must map to different digits
- The equation must be satisfied numerically.

We solve it using **Backtracking + Constraint Propagation**.

Let's assign values that satisfy:

markdown

```
  9567
+ 1085
-----
10652
```

SPPU-TE-COMP-CONTENT – KSKA Git

So, the mapping is:

Letter	Digit
S	9
E	5
N	6
D	7
M	1
O	0
R	8
Y	2

Verification:

```
diff
SEND = 9567
+ MORE = 1085
-----
MONEY = 10652 ✓
```

➤ NOV / DEC 2023

Q1)

a) List All problem solving strategies. What is backtracking, explain with n queen problem. [8]

Problem-Solving Strategies in AI:

1. Brute Force Search
2. Heuristic Search
3. Greedy Search
4. Backtracking
5. Branch and Bound

6. Constraint Satisfaction
 7. Local Search (e.g., Hill Climbing)
 8. Adversarial Search (e.g., Minimax)
-

What is Backtracking?

Backtracking is a **depth-first search-based** problem-solving technique that incrementally builds a solution and abandons (backtracks) as soon as it determines the current path won't lead to a valid solution.

It is used in constraint-based problems like Sudoku, N-Queens, etc.

N-Queen Problem Using Backtracking:

Problem: Place N queens on an N×N chessboard such that no two queens attack each other (no two in the same row, column, or diagonal).

Approach:

1. Start from the first row and place a queen in each column.
2. For each valid placement, move to the next row.
3. If placement causes conflict, **backtrack** and try next column.
4. Repeat until solution is found or all possibilities are exhausted.

Example (4-Queen Problem):

Saved memory full ⓘ

java

Copy

Edit

Initial board (empty):

```

- - - -
- - - -
- - - -
- - - -

```

Step-by-step:

- Place Q at (0, 1)
- Place Q at (1, 3)
- Conflict at (2, 0), backtrack
- Try Q at (2, 2)
- Place Q at (3, 0) => Solution found

Final board (one solution):

css

Copy

Edit

```

- Q - -
- - - Q
- - Q -
Q - - -

```

- b) Write Minimax Search Algorithm for two players. How use of alpha and beta cut-offs will improve performance? [9]

Minimax Algorithm (for Two Players)

The **Minimax algorithm** is used in **adversarial games** (like chess, tic-tac-toe) to find the optimal move by **assuming both players play optimally**.

- **MAX player** tries to maximize the score.
- **MIN player** tries to minimize the score.

Algorithm (Pseudocode):

```
python
function MINIMAX(node, depth, maximizingPlayer):
    if node is terminal or depth == 0:
        return heuristic value of node

    if maximizingPlayer:
        maxEval = -∞
        for each child of node:
            eval = MINIMAX(child, depth - 1, False)
            maxEval = max(maxEval, eval)
        return maxEval
    else:
        minEval = +∞
        for each child of node:
            eval = MINIMAX(child, depth - 1, True)
            minEval = min(minEval, eval)
        return minEval
```

Alpha-Beta Pruning

Alpha-Beta pruning is an **optimization technique** for the Minimax algorithm. It prunes (cuts off) branches that **won't affect the final decision**, reducing the number of nodes evaluated.

Key Terms:

- **Alpha (α):** Best value that the **maximizer** can guarantee so far.
- **Beta (β):** Best value that the **minimizer** can guarantee so far.

Condition to Prune:

If $\beta \leq \alpha$, stop exploring that branch.

Improvement Due to Alpha-Beta Cut-Offs:

Without Alpha-Beta	With Alpha-Beta
Time complexity: $O(b^d)$	Best case: $O(b^{d/2})$
All nodes are explored	Useless branches are skipped
Slower	Faster

Example:

In a tree of depth 3 with branching factor 2,

- Minimax explores all 8 nodes.
- Alpha-Beta may explore only 5, improving speed.

Q2

a) Define Game Theory, Differentiate between Stochastic and Partial Games with Examples. [9]

1. Game Theory (Definition):

Game theory is a branch of mathematics and artificial intelligence that studies **strategic interactions** between **two or more rational agents** (players).

It helps in **decision-making** where the outcome depends not only on an agent's actions but also on the actions of others.

Used in **AI for adversarial games**, economics, and multi-agent systems.

Feature	Stochastic Game	Partially Observable Game
Definition	A game that includes random events or chance nodes	A game where players have limited or incomplete information
Uncertainty Source	Due to randomness (dice, card draw, etc.)	Due to lack of full observability
Environment	Non-deterministic	May be deterministic but not fully known to players
AI Handling	Requires probability modeling and expectimax search	Requires belief states or partially observable MDPs
Example	Backgammon (dice roll), Poker (shuffling)	Poker (hidden cards), Battleship, Wumpus World

b) Define is Constraint satisfaction problem, State the types of consistencies Solve the following Crypt Arithmetic Problem. B A S E + B A L L = G A M E S

1. Constraint Satisfaction Problem (CSP):

A **Constraint Satisfaction Problem** is a mathematical problem defined by:

- **Variables:** A set of variables $X = \{X_1, X_2, \dots, X_n\}$ $X = \{X_1, X_2, \dots, X_n\}$
- **Domains:** Each variable has a domain D_i of possible values
- **Constraints:** A set of constraints that specify allowable combinations of values

CSP aims to assign values to variables such that **all constraints are satisfied**.

2. Types of Consistencies:

1. **Node Consistency:**
Every variable satisfies its **unary constraints** (constraints on a single variable).
 2. **Arc Consistency:**
For every value of variable XXX, there exists a **consistent value** for variable YYY such that the **binary constraint** between them is satisfied.
 3. **Path Consistency:**
Consistency between **three variables**. For every pair (X,Y)(X, Y)(X,Y), there exists a value for ZZZ such that all binary constraints are satisfied.
-

3. Solve: Cryptarithmic Problem

BASE + BALL = GAMES

Let's assign unique digits to each letter such that the sum is valid.

We assume:

- Each letter represents a **unique digit from 0 to 9**.
- No number can start with 0 (i.e., B and G \neq 0).

Let's solve it using constraint satisfaction manually.

Variables:

B, A, S, E, L, G, M

Step-by-step mapping (one valid solution):

```
CSS                                                                    Copy Edit

  B  A  S  E
+ B  A  L  L
-----
G  A  M  E  S
```

Assign digits:

Letter	Digit
B	7
A	4
S	8
E	3
L	5
G	1
M	9

Let's check:

markdown

Copy

```
7483
+ 7455
-----
14938
```

So:

- BASE = 7483
- BALL = 7455
- GAMES = 14938 ✓

- ✓ All letters map to **unique digits**
- ✓ Sum is correct
- ✓ No leading zero



A **CSP** involves assigning values to variables under constraints.

Types of consistencies: **Node, Arc, Path**

BASE + BALL = GAMES has a valid mapping:

BASE = 7483, BALL = 7455, GAMES = 14938

➤ **MAY / JUN 2024**

FULLY REPEATED QUESTION PAPER !!

➤ **NOV / DEC 2024**

Q1

a) Define Game Theory. What are the components, defining game as a Search Problem? Draw a Game Tree for tic-tac-toe. [9]

Game Theory:

Game theory is the study of mathematical models of strategic interaction between rational decision-makers. It is used in AI for decision-making in competitive environments.

Components of Game Theory:

1. Players – Participants in the game (e.g., Player X and Player O).
2. Initial State – Starting point of the game.
3. Successor Function – Rules for moving from one state to another.
4. Terminal States – End states with defined outcomes (win/loss/draw).
5. Utility Function – Numeric value for each terminal state (e.g., +1 for win, -1 for loss).

Game as a Search Problem:

- Games are modeled as trees, where:
 - Nodes = game states
 - Edges = legal moves
- The root is the initial state; branches represent possible moves.

- AI uses algorithms like Minimax to choose the best move.

Tic-Tac-Toe Game Tree (Initial Levels):

```
pgsql Copy Edit

Level 0:      (Empty Board)
              [  |  |  ]
              [---+---+---]
              [  |  |  ]

Level 1: Player X chooses one of 9 cells → generates 9 child nodes

Level 2: Player O plays on remaining 8 cells → generates more nodes

And so on, until terminal states are reached.
```

Q1 b) What do you understand by Constraint Propagation? Write short notes on Node Consistency and Arc Consistency. [8 Marks]

Constraint Propagation:

Constraint Propagation is a **technique used in Constraint Satisfaction Problems (CSPs)** to reduce the domains of variables by **enforcing constraints** before performing an exhaustive search. It helps in detecting inconsistencies early and makes the solution process more efficient.

The idea is to **propagate the effects of constraints** across the network whenever a variable's domain is modified, which may cause other variables to be updated.

Example:

In a map-coloring CSP, if a region A is assigned "Red", then neighboring regions (like B and C) cannot be "Red". So we eliminate "Red" from their domains — this is constraint propagation.

Node Consistency:

A variable is **node consistent** if all values in its domain satisfy the **unary constraints** (i.e., constraints involving only one variable).

Definition:

A node (variable) X is node consistent if **for every value v in Domain(X), the unary constraint C(X) is satisfied**.

Example:

- Let variable $X \in \{1,2,3,4,5\}$
- Unary constraint: $X > 3$
- After enforcing node consistency:
 - Reduced domain: $X \in \{4,5\}$

This reduces the search space and eliminates invalid values early.

Arc Consistency:

Arc consistency deals with **binary constraints** between two variables. It ensures that **for every value of one variable, there exists a consistent value in the connected variable**.

Definition:

A variable X is arc-consistent with variable Y if **for every value x in $\text{Domain}(X)$, there exists some value y in $\text{Domain}(Y)$** such that the binary constraint $C(X, Y)$ is satisfied.

Example:

- Variables: $X \in \{1,2,3\}$, $Y \in \{2,3\}$, Constraint: $X < Y$
- Check each value in X :
 - $X=1$: valid with $Y=2,3$
 - $X=2$: valid with $Y=3$
 - $X=3$: no value in Y satisfies $3 < Y \rightarrow$ remove 3 from X

After Arc Consistency:

- $\text{Domain}(X) = \{1,2\}$
- $\text{Domain}(Y) = \{2,3\}$

This pruning helps in eliminating inconsistent values.

Importance of Constraint Propagation:

- **Reduces the search space** by eliminating invalid values early.
- Improves **efficiency** of backtracking algorithms.
- Used in algorithms like **AC-3 (Arc Consistency Algorithm 3)**.

Q2 a) Explain how Minimax and alpha-beta algorithms change for two-player, non zero-sum games in which each player has his or her own utility function. [9]

Two-player Non-Zero Sum Game:

- In **non-zero-sum** games, one player's gain is **not equal** to the other's loss.
- Each player has **separate utility functions**.

Modified Minimax for Non-Zero-Sum:

- Minimax cannot directly apply, because we **can't minimize opponent's utility** blindly.
- Instead, each node stores a **pair of utilities (U1, U2)** for Player 1 and Player 2.
- At each turn, the current player selects the move that **maximizes their own utility**, not necessarily minimizing the other.

Alpha-Beta in Non-Zero-Sum:

- Standard alpha-beta pruning works for **zero-sum** (opponent's loss is your gain).
 - In non-zero-sum games, pruning is less effective since **utility trade-offs** are possible.
 - Modified pruning considers **dominance**: prune branches only if they are **worse for both** players.
-

Q2 b) Define Constraint Satisfaction Problem. Explain Map Coloring Example Problem. Formulate the Map Coloring Problem as CSP. [8]

CSP Definition:

A **Constraint Satisfaction Problem (CSP)** is defined by:

- **Variables:** Set of unknowns
- **Domains:** Possible values for each variable
- **Constraints:** Rules limiting allowable combinations

Map Coloring Example:

- Objective: Assign colors to regions on a map so that **no two adjacent regions** have the same color.

Example:

Coloring **3 regions**: A, B, C

- **Variables:** A, B, C
- **Domains:** {Red, Green, Blue}
- **Constraints:**
 - $A \neq B$
 - $B \neq C$

- $A \neq C$

CSP Formulation:

mathematica

Variables: A, B, C

Domains: $D(A) = D(B) = D(C) = \{R, G, B\}$

Constraints:

- $A \neq B$
- $B \neq C$
- $A \neq C$